CS 537
Lecture 15
Journaling File Systems

Michael Swift

UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

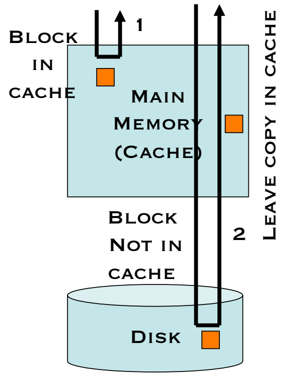
Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

Journaling File Systems

Questions answered in this lecture:
- Why is it hard to maintain on-disk consistency?
- How does the FSCK tool help with consistency?
- What information is written to a journal?
- What 3 journaling modes does Linux ext3 support?

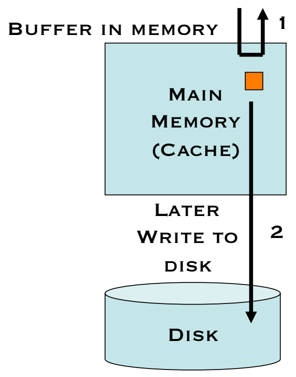
Review: The I/O Path (Reads)

- Read() from file
  - Check if block is in cache
  - If so, return block to user [1 in figure]
  - If not, read from disk, insert into cache, return to user [2]



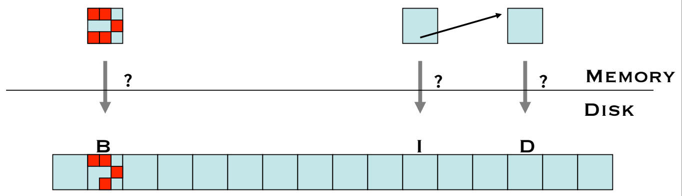
Review: The I/O Path (Writes)

- Write() to file
  - Write is buffered in memory ("write behind") [1]
  - Sometime later, OS decides to write to disk [2]
- Why delay writes?
  - Implications for performance
  - Implications for reliability



Many "dirty" blocks in memory: What order to write to disk?

- Example: Appending a new block to existing file
  - Write data bitmap B (for new data block), write inode I of file (to add new pointer, update time), write new data block D

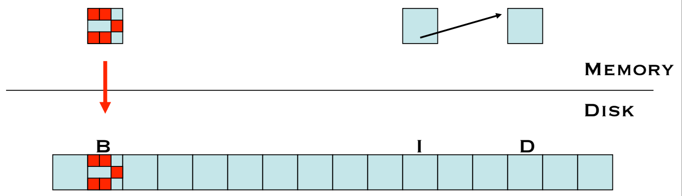


The Problem

- Writes: Have to update disk with N writes
  - Disk does only a single write atomically
- Crashes: System may crash at arbitrary point
  - Bad case: In the middle of an update sequence
- Desire: To update on-disk structures atomically
  - Either all should happen or none

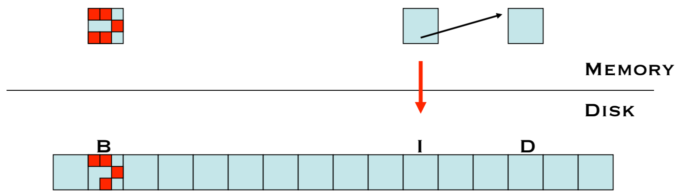
Example: Bitmap first

- Write Ordering: Bitmap (B), Inode (I), Data (D)
  - But CRASH after B has reached disk, before I or D
- Result?



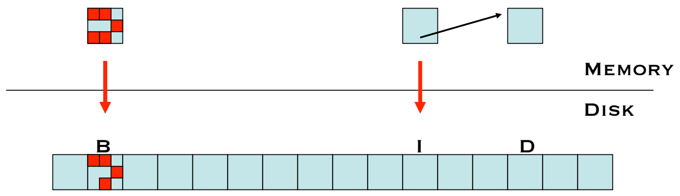
Example: Inode first

- Write Ordering: Inode (I), Bitmap (B), Data (D)
  - But CRASH after I has reached disk, before B or D
- Result?



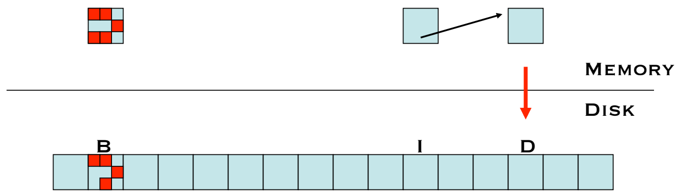
Example: Inode first

- Write Ordering: Inode (I), Bitmap (B), Data (D)
  - CRASH after I AND B have reached disk, before D
- Result?



Example: Data first

- Write Ordering: Data (D) , Bitmap (B), Inode (I)
  - CRASH after D has reached disk, before I or B
- Result?



Traditional Solution: FSCK

- FSCK: "file system checker"
- When system boots:
  - Make multiple passes over file system, looking for inconsistencies
    - e.g., inode pointers and bitmaps, directory entries and inode reference counts
  - Either fix automatically or punt to admin
  - Does fsck have to run upon every reboot?
- Main problem with fsck: Performance
  - Sometimes takes hours to run on large disk volumes

How To Avoid The Long Scan?

- Idea: Write something down to disk before updating its data structures
  - Called the "write ahead log" or "journal"
- When crash occurs, look through log and see what was going on
  - Use contents of log to fix file system structures
  - The process is called "recovery"

Case Study: Linux ext3

- Journal location
  - EITHER on a separate device partition
  - OR just a "special" file within ext2
- Three separate modes of operation:
  - Data: All data is journaled
  - Ordered, Writeback: Just metadata is journaled
- First focus: Data journaling mode

Transactions in ext3 Data Journaling Mode

- Same example: Update Inode (I), Bitmap (B), Data (D)
- First, write to journal:
  - Transaction begin (Tx begin)
  - Transaction descriptor (info about this Tx)
  - I, B, and D blocks (in this example)
  - Transaction end (Tx end)
- Then, "checkpoint" data to fixed ext2 structures
  - Copy I, B, and D to their fixed file system locations
- Finally, free Tx in journal
  - Journal is fixed-sized circular buffer, entries must be periodically freed

What if there' s a Crash?

- Recovery: Go through log and "redo" operations that have been successfully commited to log
- What if …
  - Tx begin but not Tx end in log?
  - Tx begin through Tx end are in log, but I, B, and D have not yet been checkpointed?
  - What if Tx is in log, I, B, D have been checkpointed, but Tx has not been freed from log?
- Performance? (As compared to fsck?)

Complication: Disk Scheduling

- Problem: Low-levels of I/O subsystem in OS and even the disk/RAID itself may reorder requests
- How does this affect Tx management?
  - Where is it OK to issue writes in parallel?
    - Tx begin
    - Tx info
    - I, B, D
    - Tx end
    - Checkpoint: I, B, D copied to final destinations
    - Tx freed in journal

Problem with Data Journaling

- Data journaling: Lots of extra writes
  - All data committed to disk twice
    (once in journal, once to final location)
- Overkill if only goal is to keep metadata consistent
- Instead, use ext2 writeback mode
  - Just journals metadata
  - Writes data to final location directly, at any time
- Problems?
- Solution: Ordered mode
  - How to order data block write w.r.t. Tx writes?

Conclusions

- Journaling
  - All modern file systems use journaling to
    reduce recovery time during startup
    (e.g., Linux ext3, ReiserFS, SGI XFS, IBM JFS, NTFS)
  - Simple idea: Use write-ahead log to record some
    info about what you are going to do before doing it
  - Turns multi-write update sequence into a single
    atomic update ("all or nothing")
  - Some performance overhead: Extra writes to journal
    - Worth the cost?