6.824 2024 Midterm Exam: Crash Recovery in File Systems

Time: 120 minutes Instructions: - You may consult 6.824 lecture notes, papers, and lab code. - If you have clarifying questions, ask them in a **private** Piazza post to the staff. - Do **not** discuss the exam with anyone. - **Show all your work** for full credit. Partial credit will be given for reasoning, even if the final answer is incorrect. - **Assume** a UNIX-like file system unless otherwise specified.

Section A: Short Answer (30 points)

Answer each question concisely (1–3 sentences).

1. **(5 pts)** The slides describe a "delete/create inconsistency" where an `unlink("f1")` followed by a `create("f2")` can lead to undetected corruption after a crash. Briefly explain **why** this inconsistency is *undetected* by `fsck`.
2. **(5 pts)** What is the **key invariant** that UNIX file systems aim to preserve during crash recovery regarding directory entries and i-nodes? State it precisely.
3. **(5 pts)** Why does the UNIX `fsck` program sometimes ask the user for input during recovery? Give an example scenario where this might happen.
4. **(5 pts)** The slides mention that synchronous writes in file creation can be reduced by deferring some operations. Which **two writes** (from the 7-step `create` sequence) are **most critical** to perform synchronously, and why?
5. **(5 pts)** Explain why a write-back disk cache improves performance for file creation but introduces crash recovery challenges. Use the `create("d/f")` example from the slides in your answer.
6. **(5 pts)** Suppose a file system **never** writes the free block bitmap to disk (step #1 and #5 in `create`). How would `fsck` recover the correct free list after a crash? Would this approach work for **all** crash scenarios? Justify your answer.

Section B: Problem Solving (40 points)

Work through each scenario carefully. Show your reasoning.

B1. Crash Recovery Scenarios (20 pts)

Consider the following sequence of operations on a UNIX file system with a write-back cache: `c fd = create("dir/file1", 0666); write(fd, "abcde", 5); unlink("dir/file1"); fd = create("dir/file2", 0666); write(fd, "fghij", 5);` The disk writes for these operations (from the slides) are: - `create("file1")`: 1, 2, 3, 4, 5, 6, 7 - `unlink("file1")`: 8, 9, 10, 11 - `create("file2")`: 1, 2, 3, 4, 5, 6, 7

Assume the cache flushes **only the following writes** to disk before a crash: - For `file1`: writes 1, 2, 3, 6 - For `unlink("file1")`: write 8 - For `file2`: writes 1, 2, 3

After recovery: 1. **(5 pts)** What does the file system tree look like? List the visible files in `dir/` and their contents (if any). 2. **(5 pts)** Which i-nodes and blocks (if any) are **leaked** (allocated but unreachable)? How would `fsck` handle them? 3. **(5 pts)** Is this a **benign** outcome (i.e., could it have resulted from a crash at a slightly earlier or later time)? Why or why not? 4. **(5 pts)** Propose a **minimal set of synchronous writes** (from the original 7-step sequence) that would prevent this inconsistency. Justify your choice.

B2. `rename()` Crash Recovery (20 pts)

The slides briefly mention `rename()` but do not detail its crash recovery challenges. Suppose `rename("a/b", "a/c")` performs the following disk writes: 1. Update the i-node for `b` (if its name is stored there). 2. Remove the directory entry `"b"` from `a`'s contents. 3. Add the directory entry `"c"` to `a`'s contents. 4. Update `a`'s i-node (mtime, length).

Assume a crash occurs after **only step 2** completes.

1. **(5 pts)** What does the file system look like after recovery? Is `b` or `c` visible? What happens to `b`'s i-node and data?
2. **(5 pts)** Could `fsck` automatically repair this? If not, what would it ask the user?
3. **(5 pts)** Propose a **synchronous write order** for `rename()` that guarantees recovery to a consistent state. Hint: Think about commit points.
4. **(5 pts)** How would your answer change if `b` and `c` were in **different directories** (e.g., `rename("a/b", "x/c")`)? List the additional challenges.

Section C: Design (30 points)

Apply concepts from the slides to design solutions.

C1. Optimizing `create()` (15 pts)

The slides suggest that only **two synchronous writes** (steps 2 and 6) are strictly necessary for correct `create()` recovery, but UNIX synchronously writes steps 2 and 3.

1. **(5 pts)** Why might UNIX choose to synchronously write step 3 (`d`'s contents) even if it's not strictly required for correctness? Hint: Consider `fsck`'s behavior.
2. **(5 pts)** Suppose we **defer step 3** (as the slides suggest) but still synchronously write step 2. Describe a crash scenario where this leads to a **user-visible inconsistency** (e.g., a file appears to exist but cannot be opened).
3. **(5 pts)** Propose a **hybrid approach** that reduces synchronous writes while avoiding the inconsistency in (2). Your solution may involve additional metadata or reordering.

C2. Journaling vs. `fsck` (15 pts)

The slides focus on `fsck`-based recovery, but modern file systems often use **journaling** (e.g., ext4, XFS).

1. **(5 pts)** How would a **journaling file system** (e.g., with metadata journaling) handle the `unlink("f1")` + `create("f2")` inconsistency from the slides? Would it detect the issue? Why or why not?
2. **(5 pts)** Journaling introduces **write overhead**. For the 7-step `create()` sequence, which steps would be **journalled**, and which could be written directly to their final locations? Justify your answer.
3. **(5 pts)** A critic argues: *"Journaling is unnecessary because `fsck` can always recover the file system to a consistent state."* Write a **counterargument** using examples from the slides (e.g., undetected inconsistencies or performance trade-offs).

Section D: Essay (20 points)

*Choose **one** of the following. Write a clear, structured response (1–2 paragraphs).*

1. **Performance vs. Durability Trade-offs** The slides emphasize that crash recovery must be **fast** because persistent storage is slow. Discuss how the following techniques balance this trade-off:

    - Write-back caching
    - Synchronous writes for commit points
    - `fsck` vs. journaling Use examples from the slides (e.g., `create()` throughput) to support your argument.

2. **Distributed File Systems** The slides note that crash recovery in disk file systems shares similarities with distributed systems. Compare and contrast:

    - How a **single-machine file system** (e.g., UNIX) handles recovery after a crash.
    - How a **distributed storage system** (e.g., Spanner or FaRM from previous exams) handles recovery after a node failure. Focus on **ordering guarantees**, **commit protocols**, and **user-visible inconsistencies**.

End of Exam *Good luck! Remember to show all your work.*