

CS167 Homework Assignment 3

Due November 13, 2009

1. The text, starting on page 7-35, discusses a technique known as soft updates. It mentions on page 7-38 that FFS using soft updates may require more disk writes than Async FFS. Give an example in which more disk writes would occur with soft updates than with Async FFS.
2. NTFS uses a combination of both redo and undo journaling. Explain why both are used (and not just one).
3. One feature of RAID-Z is that the stripes are of variable width — each block has at least one parity sector associated just with it. However, to determine what the block boundaries are, one must traverse the file-system metadata (the shadow-page tree). In a RAID-4 system, one knows that parity information resides exclusively on one disk. Suppose that we lose a disk in a RAID-4 system and a RAID-Z system.
 - a. Explain what must be done in each case to recover.
 - b. Under what circumstances is recovery faster in the RAID-4 system? (No need for exact details — just explain the principle.)
4. An issue with ZFS is reclamation of data blocks. With multiple snapshots in use for each file system, it's not necessarily trivial to determine which blocks are actually free. For example, there might be, in addition to the current version of the file system, three snapshots of it. If we delete the snapshot that's neither the most recent nor the least recent, how can we determine which blocks must be deleted and which must remain? To answer this question, let's go through the following steps.
 - a. Explain why reference counts are not part of the solution. (Hint: consider efficiency.)
 - b. Suppose each block pointer, in addition to containing a checksum, also contains the time at which the block was created, known as the block's *birth time* (note that blocks are never modified in place, thus the creation time for a particular file-system block on disk doesn't change). Explain how this information can be used to determine that a block was created after a particular snapshot. (Hint: might timestamps be associated with other things as well?)
 - c. Suppose a block is freed from the current file system (but might still be referenced by a snapshot). Explain how it can be determined whether there exists a snapshot that is referring to it.
 - d. If a block being freed cannot be reclaimed because some snapshot is referring to it, a reference to it is appended to the file-system's *dead list*. When a new snapshot is created, its dead list is set to be that of the current file system, and the current file-system's dead list is set to empty. Let's assume no snapshots have ever been deleted. We say that a snapshot is *responsible* for the existence of a block if the block's birth time is later than the birth time of the previous snapshot, and the block was freed before the birth time of the next snapshot (or of the file system if this is the most recent snapshot). In other words, the only reason the block cannot be reclaimed is because this snapshot exists. Are all blocks for which a snapshot is responsible listed in the next snapshot's dead list (or in the current file system's dead list if this is the most recent snapshot)? Explain.
 - e. Suppose a snapshot is deleted (the first one to be deleted). How can we determine which blocks to reclaim?

- f. We'd like this reclamation algorithm to work for subsequent deletions of snapshots. Part d describes an invariant: all blocks for which a snapshot is responsible appear in the next snapshot's dead list (or in the dead list of the file system if it's the most recent snapshot). What else must be done when a snapshot is deleted so that this invariant is maintained (and the algorithm continues to work)?