CS167 Homework Assignment 3

Due 11:59pm April 6, 2018

1. [40%] 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. Note that putting reference counts on file-system blocks is not part of the solution: if reference counts were used, then each block's reference count would have to be incremented on each snapshot operation — greatly adding to the cost of creating the snapshot.
   a. [8%] 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 snapshots as well?)
   b. [8%] 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.
   c. [8%] 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.
   d. [8%] Suppose a snapshot is deleted (it is the first one to be deleted). How can we determine which blocks to reclaim?
   e. [8%] 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)?
2. [20%] Section 7.3.2.2 of the textbook discusses virtual-memory management in Windows. It mentions a potential deadlock situation in which one of the two modified-page-writer threads invokes pageable file-system code.
   a. [10%] Describe a scenario in which such a deadlock might happen (i.e., there is only one modified-page writer thread).
   b. [10%] Explain how dividing the duties of the modified-page-writer threads — so that one handles page frames containing file data and executes pageable code and the other

handles page frames containing private data and executes non-pageable code — avoids deadlock.

3. [40%] We have a guest operating system that supports virtual memory, running in a virtual machine on a VMM that, of course, also supports virtual memory. (You may assume for this problem that we have only a single virtual machine.) Assume the real processor is an Intel x86-64 without extended page table (EPT) support. Thus it has no special support for implementing page translation on a virtual machine. Thus the guest OS constructs page tables mapping virtual-virtual memory (of its applications) to virtual-real memory. The VMM might have page tables mapping virtual-real memory to real memory. However, if the system is running an application of the guest OS (in virtual-virtual memory), the page table used by the hardware must map virtual-virtual memory to real memory.

   a. [5%] Explain how the VMM constructs this page table (the one mapping virtual-virtual memory to real memory). You do not need to go into the details of how the multi-level page table is set up, but describe how it is determined, for each page of virtual-virtual memory, what its translation into page frames of real memory is, if the translation exists. Otherwise indicate which translations are marked invalid.

   b. [35%] A possible concern with such a double implementation of virtual memory (on both the guest OS and the VMM) is that both are managing system resources, possibly in conflict with each other. One approach to avoiding such a conflict is, essentially, to put the guest OS in charge of managing memory resources. The VMM puts aside a fixed number of page frames, say N, for use by the guest OS. The mapping from virtual-real to real memory would map the first N pages of virtual-real memory to these N page frames and would not change for the life of the virtual machine running the guest OS. The guest OS would then be responsible for mapping virtual-virtual memory to this fixed amount of virtual-real memory. N, of course, would be less than the total amount of page frames on the real machine. The guest OS would thus be solely responsible for determining which virtual-virtual pages are currently mapped to real memory.

      Alternatively, management of memory resources could be handled by the VMM. The size of virtual-real memory could be made large enough so that there are enough virtual-real pages so that all active virtual-virtual pages could be mapped one-one to virtual-real pages. However, the VMM would now have to support mapping this large virtual-real address space onto a much smaller real address space. Thus it's the VMM that determines which virtual-virtual pages are currently mapped to real memory.

      Let's assume the real computer has 2^{32} bytes of real memory. For the first approach, assume it assigns 2^{30} bytes to the virtual machine running the guest OS. For the second approach, assume the virtual-real size is 2^{48} bytes. Assume the page size is 2^{12} bytes.

      What are the advantages of the first approach? What are the advantages of the second approach? Which do you think makes the most sense? Explain.