CS167 Homework Assignment 2 Solutions

Spring 2018

1. *As discussed in class, virtual machines can be nested, i.e., a virtual machine monitor can run on a virtual machine. In what follows we say that the level-0 VMM run directly on the real hardware. A level-1 VMM runs in a virtual machine (VM) of the level-0 VMM. In general, a level i VMM runs in a VM of the level i-1 VMM. In this problem we consider how virtual machine monitors designed for Intel's VT-x enhancement to the x86 architecture can run on virtual machines. A rough description of how VT-x works is that the machine runs either in root mode (in ring -1) or in non-root mode (in rings 0-3). The VMM runs in root mode, its virtual machines (VMs) run in non-root mode. When in non-root mode, certain actions cause VMexits, which result in traps to root mode. The intent is that these VMexit-causing actions are those, such as modifying certain important registers or modifying certain memory locations, that would affect other VMs. The VMM specifies, for each of its virtual machines, which events cause VMexits.*

   a. *[5%] An "off-the-shelf" VMM for the VT-x architecture runs in hardware ring -1 and provides a virtual environment that appears to be an x86 with support for rings 0 through 3 only. Explain why such a VMM does not support nested virtualization. (The expected answer is short.)*

      The virtual machine supported by the VMM is one without VT-x (since it doesn't support ring -1). Since the VMM requires VT-x, it cannot run in the virtual machine and thus cannot be nested.

   b. *[10%] As part of switching to non-root mode to run a virtual machine, the VMM specifies which actions by the VM trigger VMexits (traps to the VMM). Thus, in principle, each VM may have a different set of actions that trigger VMexits. When virtualization is nested, VMexit traps cause the bottom-level (level-0) VMM to be invoked. Since only the bottom-level VMM is running in real ring -1, only it can switch to a VM and thus establish which actions by the VM cause VMexits. Suppose a VMM at level 1 (i.e., running in a VM of the level-0 VMM) attempts to switch to one of its VMs, and thus specifies which actions cause VMexits. The action of switching to a VM will have been set up by the level-0 VMM to cause a VMexit, and thus the action causes a trap to ring -1, which is handled by the level-0 VMM. Thus it's the level-0 VMM that actually tells the hardware which actions cause VMexits. What set of actions does it specify as causing VMexits? (The choices would be those actions specified by the level-1 VMM for its VM, those specified by the level-0 VMM for its VM, the union of these two sets of actions, or the intersection of these two sets of actions.)*

      When the VM supported by the level-1 VMM is running, both the events that the level-1 VMM wants to cause *VMexits* and the events that the level-0 VMM wants to cause *VMexits* must be set up to cause *VMexits* to the level-0 VMM.

   c. *[10%] In general, when a level-i VMM, for i>1, switches to one of its VMs, must the level-0 VMM invoke the level-1 VMM, which in turn invokes the level-2 VMM, etc., or can the level-0 VMM start the level-i VM directly?*

      When a level-i VMM switches to one of its VMs, a *VMexit* occurs that's handled (as they all are) by the level-0 VMM. This VMM then directly starts the VM, using the union of the *VMexit*-causing events requests by all VMMs up through the level-i VMM.

2. *A standard feature on many x86-based systems has been a programmable interval timer (PIT). As used by operating systems, it is given an initial positive value. It then counts down at some*

frequency (say 1 MHz) until it reaches zero. When this happens, it sends a clock interrupt to the processor, resets itself to the initial value, and counts down again, ad infinitum. For the following questions, you may assume a uniprocessor system. The expected answers are relatively short.

a. *[12%] Assume that the PIT's initial value is set just once, when the system boots. Explain how the PIT might be used by an operating system to drive a time-slice-based scheduler, where each thread is given a time slice of one centisecond (hundredth of a second), and clock interrupts happen every millisecond (thousandth of a second).*

   One approach is for the clock interrupt to maintain a 64-bit (so that overflow is not a problem) global counter that starts at zero and is incremented by one at every clock interrupt. When a thread starts its time slice, the global-timer value at which its time slice will be over is computed. Once the clock-interrupt handler sets the counter to that value, the thread is forced to yield the processor to the next thread.

b. *[13%] Our operating system is now running on a virtual machine. We would like its threads to get one-centisecond time slices of virtual time, i.e., even though several centiseconds of real time may have elapsed, the time slice is not over until the virtual machine has been running for a centisecond. Explain what is done on the virtual machine monitor (VMM) to ensure that this happens. Note that it's not necessary for these time slices to be measured exactly, as long as they are on average the correct length (with low variance).*

   The VMM's clock-interrupt handler is notified of clock interrupts every millisecond of real time. If the VM is running when the clock interrupt occurs, the VMM simulates the occurrence of a clock interrupt on the VM by saving its state on the VM's current kernel stack and invoking its clock-interrupt handler.

3. *Assume we're using the Rhinopias disk drive.*

   a. *[8%] Suppose we are using S5FS and we've doubled the block size from 512 bytes to 1024 bytes. Is the maximum expected transfer rate roughly doubled? Explain.*

      Yes. Since the blocks of a file are randomly scattered, fetching each requires an average seek time plus an average rotational delay. The time for the actual transfer of a block is negligible in comparison. By doubling the block size, after each seek and rotational delay, twice as much data is transferred as before.

   b. *[8%] We've done the same with FFS. Is the maximum expected transfer rate roughly doubled? Explain. Assume that we're using two-way block interleaving: when we allocate successive blocks on a track, one block is skipped. Thus, in the best case, every other block of a track is allocated to a file.*

      In the best case, in which the blocks of a file are all within a cylinder group and arranged on tracks has efficiently as is permitted, each rotation of the disk causes half a track's worth of data to be transferred. This is unaffected by the block size.

   c. *[9%] Why should we care about the block size in FFS?*

      Consider the case in which FFS is being used on a busy server, with concurrent requests for blocks from many files. The situation could be as bad as things are with S5FS, in which successive requests are to random disk locations. Thus the analysis given in part a for the speedup in S5FS applies.

4. *[25%] Explain how renaming a file can be done using the consistency-preserving approach so that a crash will not create situation in which the file is lost. Be sure to take into account the link*

count stored in the file’s inode. Note that there may have to be a temporary inconsistency; if so, explain why it will not result in lost data.

As explained in the text, to rename a file you should first create a link to the file under its new name, then remove the link corresponding to its old name. Creating the new link cannot be done atomically, since both the directory must be updated to contain the link and the file’s inode must be updated to increase its link count from 1 to 2. If the directory entry is created before the link count is updated, there could be a problem if the system crashes between the two operations. After the crash, though there are two links to the file, its link count is one. If one of the links is removed, the link count is reduced to zero and the file might be deleted since it appears to be no longer referenced. On the other hand, if the link count is updated before the directory entry is created, and a crash occurs between the two operations, there will be just one entry referring to the file, even though it has a link count of two. The worst that could happen in this case is that someone intends to delete the file by removing that link, but because it would still have a positive link count, the file continues to exist, but is not referenced by any directory. This, as explained in the text, is an innocuous inconsistency.

After the new link is created, we have a similar problem in removing the old link. Should the link be removed first, then the file’s link count decremented by one, or vice versa? Following reasoning similar to the above, we should first remove the old link, then decrement the file’s link count, so that the link count is always at least as great as the actual number of links.