Virtualization and Cloud Computing

Practice Problems

Short Answer Questions

1. Consider an x86 CPU that executes the `int n` instruction to process traps.

   (a) How does the CPU locate the interrupt descriptor table (IDT) entry corresponding to this particular interrupt?

   Ans: The IDTR register has IDT base address, and the interrupt vector 'n' is used as index into IDT.

   (b) The IDT entry specifies, among other things, values to write into certain CPU registers before the kernel can handle the interrupt. State any two CPU registers whose values are specified by an IDT entry.

   Ans: CS, SS, EIP, ESP

2. Consider an x86 CPU that uses a combination of segmentation and paging to translate virtual memory addresses to physical addresses.

   (a) Suppose the CPU tries to fetch an instruction from the virtual address location stored in the EIP register. How is this virtual address converted into a physical address using a combination of segmentation and paging?

   Ans: Base of CS segment + offset (which is EIP value) = Linear Address ; Translate this linear address using page table to get physical address.

   (b) Name one way by which the value of the CPU's `cs` register can be changed.

   Ans: Interrupt / syscall / context switch

   (c) Name one way by which the value of the CPU's `ds` register can be changed.

   Ans: Mov an index into ds to write the corresponding segment descriptor into ds

   (d) Name two CPU registers whose last 2 bits capture the current privilege level of the CPU.

   Ans: CS, SS

3. Suppose you are trying to build a trap-and-emulate VMM to virtualize an OS on the x86 architecture, using a technique like binary translation. The VMM runs the guest OS at a lower privilege level, while it runs itself at the highest privilege level of ring 0. Now, while the CPU instruction that writes to the `idtr` register (interrupt descriptor table base register) is a privileged instruction, the instruction that reads the `idtr` is an unprivileged instruction that executes correctly in any privilege level without trapping to ring 0. Should the VMM replace the instruction to read the `idtr` by something that traps to the VMM (and can be emulated by the VMM) during binary translation? Answer yes/no and justify. If you answer yes, explain what would go wrong if this instruction is executed directly by a virtualized guest without trapping to the VMM. If you answer no, explain why this instruction is safe to execute by a virtualized guest.

Ans: Yes, VMM should replace the instruction and not let the guest directly read the IDTR. This is because the guest OS would have replaced the IDTR, and the guest OS will panic after finding out that the value it set is not the same as the value it read from the IDTR.

4. Consider a Linux-like OS that is built to run on non-virtualized systems, and is compiled into a binary for the x86 architecture. A VMM wishes to run this OS within a guest VM using the trap-and-emulate method. The underlying hardware has no virtualization support.

   (a) Can the VMM run this unmodified OS binary as a guest directly on the hardware and achieve correct virtualization? That is, can all the instructions in the guest OS binary run directly on the CPU without any modification? Answer yes/no and justify. If you answer yes, give an example of a VMM that runs unmodified OS binaries in this manner. If you answer no, explain why this is not possible to do.

   Ans: No, Due to unprivileged sensitive instructions

   (b) Can the VMM run this OS as a guest without having to modify its source code and achieve correct virtualization? Answer yes/no and justify. If you answer yes, give an example of a VMM that runs unmodified OS source in this manner. If you answer no, explain why this is not possible to do.

   Ans: Yes, VMware workstation hypervisor / Full Virtualization / binary translation

5. Consider the QEMU/KVM VMM that implements hardware-assisted virtualization.

   (a) A QEMU process has memory mapped a guest OS image into its address space, and is scheduled by the host OS scheduler on a certain CPU core. Once the guest VM has been suitably configured using the KVM APIs, which system call(s) does QEMU use to begin the execution of the guest VM code on the CPU core?

   Ans: ioct (vcpu_fd, KVM_RUN ,..);

   (b) When QEMU invokes the system call(s) in part (a) above, which CPU instruction(s) are used by KVM to begin execution of the guest VM on the CPU core?

   Ans: VMLAUNCH / VMRESUME

6. Consider a guest VM running on the QEMU/KVM hypervisor. Assume that the guest has just exited.

   (a) How does the CPU obtain the address of the KVM code in the host to jump to upon an exit?

   Ans: VMCS

   (b) After control returns to KVM, from which memory location/data structure/CPU register does KVM obtain the reason for the VM's exit?

   Ans: VMCS

   (c) Does (and, can) the QEMU userspace process see all exits of a VM run by it? Answer yes/no and justify. If you answer yes, explain why QEMU needs to see all VM exits. If you answer no, give an example of a VM exit that is not visible in QEMU userspace.

   Ans: No, for example, Timer interrupt / Keyboard interrupt / Page fault

(d) If a VM exit needs to be handled by QEMU, from which memory location/data structure/CPU register does QEMU obtain the reason for the VM's exit?

Ans: Struct KVM_RUN

7. Consider a Linux-like OS doing interrupt-driven I/O to communicate with a hard disk. When a process makes a blocking `read` system call, the OS issues instructions to the disk to read one or more blocks, and the disk raises an interrupt when the reading is complete.

(a) Suppose the OS is running on baremetal in a non-virtualized scenario. What does the OS do from the time it issues a read to the disk to the time the disk returns the data back with an interrupt, given that the process that issued the read is blocked for the data and cannot run?

Ans: OS context switches to another process

(b) Now, suppose this OS is running as a guest within a QEMU/KVM-like hypervisor. Assume that the QEMU process running this guest has only a single thread of execution, and it emulates disk I/O on behalf of the guest by issuing system calls on the host OS. Answer the question in part (a) once again in this scenario.

Ans: Guest OS is blocked

(c) Suppose we improve the QEMU design in part (b) to make it multi-threaded. I/O operations on the host are performed by separate I/O threads within the QEMU process, and the main thread gets back to running the guest VM. However, all the QEMU threads run on a physical machine with only one CPU core. Answer the question in part (a) once again in this scenario.

Ans: Guest OS runs normally

(d) Suppose we improve the scenario in part (c) further by providing multiple CPU cores in the physical machine that runs the QEMU process. Answer the question in part (a) once again in this scenario.

Ans: Guest OS runs normally

8. Consider an architecture that uses hierarchical paging, where all page tables (regular, extended, shadow, and so on) have N levels. A hypervisor on this system runs a single guest VM, which in turn has K active processes.

(a) If the hypervisor uses extended page tables for memory virtualization, how many extra page tables does the hypervisor maintain to virtualize the guest, in addition to the page tables maintained by the guest VM itself? Also, approximately how many memory accesses must the MMU perform in order to "walk the page table" and translate an address?

Ans: 1 EPT, O(N^2) accesses

(b) Repeat part (a) above when the hypervisor uses shadow page tables.

Ans: K Shadow page tables, O(N) accesses

(c) Explain the differences (if any) in the mappings stored by the TLB in parts (a) and (b) above.

Ans: No difference (in both cases, GVA to HPA is stored)

9. Consider the VMWare Workstation hypervisor studied in class.

   (a) Does the binary translation of the guest OS code happen in the host context or the guest/VMM context?

   Ans: Guest/VMM context

   (b) At what privilege level does the VMM's binary translator logic usually run?

   Ans: Ring 0

   (c) Give an example of guest OS binary code / CPU instruction that must be replaced by the VMM during binary translation.

   Ans: Popf, writing cr2/cr3/idtr registers, setting eflags.if

   (d) In which part of the address space of the guest process does the translated binary code of the guest OS reside?

   Ans: Top/last 4MB of address space, in translation cache

   (e) What are the base/limit values of the `cs`, `ds`, and `gs` segment registers when the VMM is executing translated guest OS code? (You may show it in a picture also.)

   Ans: cs and gs: base = 4GB-4MB, limit = 4GB
   ds: base = 0, limit = 4GB-4MB

10. Consider a VMM running a guest OS on a hardware that has no EPT support for memory virtualization. That is, the hardware MMU takes a pointer to only one page table.

    (a) Is it possible that this single page table accessed by the MMU resides not in host/VMM memory, but directly in the guest VM memory? Answer yes/no and justify. If you answer yes, give an example of a system where this happens. If you answer no, explain why this cannot happen.

    Ans: Yes, XEN (paravirtualization), shadow page tables as in ELI

    (b) Is it possible that this single page table is always updated by the guest VM directly, without any intervention from the hypervisor? Answer yes/no and justify. If you answer yes, give an example of a system where this happens. If you answer no, explain why this cannot happen.

    Ans: No, HPA validation by hypervisor is required

11. Consider a page fault in a virtualized system that occurs when a guest VM is executing, causing a VM exit to the hypervisor. The faulting address was accessed by code running in the guest.

    (a) Give an example of a scenario where the page fault is handled only by hypervisor, and the guest is not made aware of it.

    Ans: Hidden page faults (eg. GPA to HPA mapping not present in EPT)

    (b) Give an example of a scenario where the page fault is injected back into the guest VM for further handling.

    Ans: True page faults (eg. GVA to GPA mapping not present in guest page table)

12. Consider the design of paravirtualized device drivers based on the concept of shared I/O rings between the guest and the hypervisor, a design found in the device drivers of Xen and the virtio drivers in QEMU/KVM.

    (a) After the guest places multiple I/O requests on the virtio queue, how is QEMU/KVM notified about the arrival of the requests?

    Ans: Via interrupts which causes VM exits. Guest notifies Qemu via "kick" which causes VM exit

    (b) After the guest domain places multiple I/O requests on the shared I/O ring, how is the driver domain in Xen notified about the arrival of the requests?

    Ans: Via hypercalls/asynchronous event channels

13. Consider the three main hypervisors studied in class: QEMU/KVM, VMWare Workstation, and Xen.

    (a) For which of these hypervisors is the OS source code written to run in a less privileged ring and not in ring 0?

    Ans: Xen

    (b) In which of these hypervisors does the binary code / CPU instructions of the guest OS execute on the CPU in ring 0?

    Ans: QEMU/KVM

14. Consider the handling of a network packet reception event in the kernel I/O stack.

    (a) When the NIC is a regular, non-SRIOV NIC and the kernel is running on baremetal (i.e., not virtualized), are the addresses of DMA buffers provided to the NIC by the kernel device driver virtual addresses or physical addresses? Justify your choice.

    Ans: Host Physical Address

    (b) When the kernel is running inside a VM, and the device driver is a VF driver that is communicating with an SRIOV VF assigned to this VM, are the addresses of DMA buffers provided to the VF by the driver guest virtual addresses or guest physical addresses or host physical addresses? Justify your choice.

    Ans: Guest Physical Address

    (c) The IOMMU within the SRIOV NIC translates addresses of type X into addresses of type Y, in order to correctly perform DMA into the guest VM memory. What are X and Y? Your answers must be in the form of host/guest virtual/physical addresses.

    Ans: X: GPA, Y: HPA

15. Consider the nested virtualization techniques described by the Turtles paper. Consider nesting at the L0, L1, and L2 levels as discussed in class.

    (a) Give one example of a field in VMCS(1,2) that must be changed before the L0 hypervisor can copy it into the corresponding field of the shadow VMCS(0,2) that is used to launch L2.

    Ans: Host Context/ CR3

(b) Suppose the hardware has EPT support but L0 chooses not to expose EPT capability to L1 for nesting, so that L1 uses shadow page tables to virtualize L2. In this case, when L0 launches L2, which are two page tables that it provides to the memory hardware for correct address translation from L2's virtual addresses to L0's physical addresses? Your answers should be in the form of SPT(X,Y) and EPT(M,N).

Ans: EPT(0,1) and SPT(1,2)

16. Consider the various memory reclamation techniques implemented by hypervisors when under memory pressure. For which of the memory reclamation techniques studied in class can the guest OS know or guess as to which parts of its physical address space are possibly not mapped to real physical memory? Explain how the guest OS can know this fact using the information available in its kernel data structures.

    Ans: Ballooning. Guest OS allocates some pinned pages to the balloon driver. These pages are used when VMM instructs balloon to inflate.

17. Consider the code of interrupt handlers within a guest OS that is running on a VMM. Among the hypervisors studied in class, give an example of a hypervisor that restricts the source code of guest interrupt handlers to not invoke any privileged instructions. Further, describe the mechanisms by which the hypervisor ensures that the interrupt handlers comply with this restriction.

    Ans: Xen, it checks that the segments loaded by the IDT entry do not specify execution in ring 0. If there are any other issues in the guest interrupt handler code, another trap (a "double fault") will occur and control will return back to Xen again

18. Optimizing VM exits is an important problem for all hypervisors. Describe one optimization to reduce the number of VM exits that software-based hypervisors like the VMWare Workstation can easily realize, but which will be harder to implement in a hardware-assisted hypervisor.

    Ans: Batching of interrupts before exiting/ interrupt emulated by VMM in VMM context itself.

19. Consider a hypervisor like QEMU/KVM or VMWare Workstation that switches between host and guest modes to run guest VM code natively on hardware. When a guest VM is running in the guest mode, the host OS context must have been saved somewhere in memory. For the VMM/CPU to be able to locate and access this context when switching back to host mode, is it necessary that the memory containing the saved host context be mapped into the address space of the guest processes somewhere (even if it cannot be directly accessed by the guest due to protection mechanisms)? Answer yes/no and justify your answer. If you answer yes, explain why such a mapping is essential to the design of VMMs that switch between modes. If you answer no, describe an alternate mechanism by which the VMM/CPU can access and restore the host OS context when it needs to switch back to host mode, despite this context not being accessible from the current address space.

    Ans: No, not necessary. H/W CPU can store a pointer to VMCS and VMCS contains a host state area where the host OS context is stored.

20. Consider a trap-and-emulate VMM, where certain privileged operations performed by the guest VM trap to (and are emulated by) the VMM. Is it necessary that such a VMM always be embedded within the address space of guest processes? Answer yes/no and justify. If you answer yes,

explain why such an embedding is essential to the design of trap-and-emulate VMMs. If you answer no, describe an alternate mechanism by which the CPU running guest code can jump to the trap handling VMM code that is not part of the current address space.

Ans: No, QEMU/KVM is not mapped into guest OS address space. On trap, CPU uses host context in VMCS to jump to trap handling in KVM.

21. Consider the trap-and-emulate hypervisor design for x86 architectures, as discussed in class.

    (a) Is it possible to run an OS that is not built for virtualization as a guest OS on a simple trap-and-emulate hypervisor, without performing any changes to its source code? (Assume that the underlying hardware has no virtualization support.) If yes, explain how this is possible with a suitable example. If not, explain why it is not possible.

    Ans: Yes, it is possible. We can do binary translation (full virtualization like in VMWare workstation) of guest OS code to translate any non-virtualizable instructions.

    (b) Is it possible to run an OS that is not built for virtualization as a guest OS on a simple trap-and-emulate hypervisor, without performing any changes to its stream of binary instructions? (Assume that the underlying hardware has no virtualization support.) If yes, explain how this is possible with a suitable example. If not, explain why it is not possible.

    Ans: No, it is not possible as x86 instruction set is not virtualization. (Another valid answer since the question does not explicitly say source code cannot be changed: Yes it is possible by changing source code / Xen / paravirtualization.)

    (c) Suppose the guest OS runs in ring 1 and traps to the VMM (which is running in ring 0) for privileged operations. Consider an unprivileged x86 instruction that reads the segment descriptors stored in the segment descriptor table. Explain what can go wrong if a guest OS is allowed to execute this instruction directly when running in ring 1.

    Ans: VMM installs shadow segment decriptor tables, i.e., guest OS segment descriptor tables are not directly used in the CPU since the guest expects to run in ring 0. If the guest is allowed to read the segment descriptor table, it will read the VMM's tables, and it can realize the inconsistency or panic due to not running in ring 0 etc.

    (d) Provide an example of a VMM design where the VMM is not mapped into the address space of guest processes, but the guest OS can still be made to exit to the VMM for certain privileged operations. In your example, how does the CPU know the location of the VMM trap handling code, even though it is not mapped into the address space?

    Ans: Yes, example is QEMU/KVM. CPU knows the location of the VMM code from the host context in VMCS.

22. Consider the dynamic binary translation scheme discussed in the VMWare workstation system in class, where basic blocks in the guest OS code are translated into compiled code fragments (CCFs) in the translation cache.

    (a) In which part of the guest address space is the translation cache located?

    Ans: In the VMM part of the address space / top 4MB

(b) When the guest OS is running, specify which instructions are executed from the original OS image, and which from the CCFs in the translation cache.

Ans: Guest OS runs entirely from the CCF in the translation cache (never from the original OS image). Reason: The entire guest translated code resides in translation cache, and guest execution does not (cannot) jump from original image to trnaslated image.

(c) Consider a basic block consisting of a sequence of regular arithmatic operations (using unprivileged instructions and general purpose registers), ending with a jump instruction to another address in the guest OS code. Which of these instructions (if any) has to be modified/translated when generating CCF, and why?

Ans: Regular arithmatic instructions need not be changed during translation, i.e., they are translated identically. Jump instruction needs to change because the address to jump to must change from an address in the original guest OS image to an address of another CCF in the translation cache.

(d) Does every trap from guest OS code to the VMM necessarily lead to a world switch from VMM context to host OS context? Answer yes or no. If you answer yes, explain why every trap must lead to a world switch. If you answer no, give an example where the VMM can handle the trap without performing a world switch.

Ans: No it is not necessary. For example, VMM need not do a world switch back to host for every I/O operation. Instead, it can wait for a batch of I/O operations and do one exit to VMM once per batch.

23. Consider a guest running on a hypervisor using shadow page tables for memory virtualization. For a certain guest user process with 4 logical pages (numbered 0, 1, 2, 3), pages 0, 1, and 2 are mapped to guest physical frames 31, 57, and 22 respectively. The logical page 3 is not assigned any physical frame by the guest OS. Now, when this guest OS is allocated memory in the host/hypervisor, guest physical frames 22 and 31 are mapped to host physical frames 432 and 781 respectively. Guest physical frame 57 is not assigned any memory in the host context.

(a) Suppose the guest user process accesses logical page 0. Describe the outcome of the memory translation when this access happens. That is, if the logical page number maps to a valid host phyiscal frame, provide the address of the host physical frame that is accessed, and explain how you arrive at the translated address. On the other hand, if the logical page number cannot be translated using the shadow page table, explain what happens on the page fault, who services it and how.

Ans: Host physical frame 781 is accessed.

(b) Repeat the above question when the guest process accesses page 1.

Ans: Page fault occurs, and page fault is handled by VMM/host.

(c) Repeat the above question when the guest process accesses page 3.

Ans: Page fault occurs, and this page fault is injected back into the guest and is handled by the guest. (After guest allocates physical frame for page 3, host can also allocate memory for the corresponding guest frame.)

(d) Among the two techniques of ballooning and uncooperative swapping to reclaim memory from guest VMs, which technique (if any) results in any changes to the guest page table mappings (from guest virtual to guest physical addresses)? Explain your answer.

Ans: Ballooning will result in changes to memory allocation within the guest OS, because guest allocates memory to the dummy balloon driver device. Uncooperative swapping will be transparent to the guest OS, and no changes will happen within the guest.

24. Consider a VMM like KVM/QEMU that virtualizes I/O performed by guest OSes via emulation.

(a) The userspace part of the hypervisor (QEMU) uses multiple I/O threads to perform blocking I/O operations. Why do we need these separate I/O threads? Explain what would go wrong if the vCPU thread in QEMU itself emulated the guest's I/O requests.

Ans: If vCPU thread is blocked for I/O, the guest OS execution also gets blocked. But if separate I/O thread is used, guest OS can resume execution (possibly of another user process) while waiting for I/O to complete.

(b) KVM usually configures VMCS in such a way that guests exit back to root mode on all interrupts, even for those interrupts that are destined to the guest itself. Explain why such a configuration is common in practice. That is, explain what would go wrong if a guest did not exit to root mode to handle interrupts.

Ans: It is safe to exit for all interrupts because there might be some interrupts that are meant for the host OS or other guest OS running on the same physical machine. So VMM inspects all interrupts and injects guest interrupts back into the guest VM.

(c) Describe one overhead associated with emulated I/O that is eliminated by using SRIOV.

Ans: SRIOV enables direct communication between I/O devices and the guest OS. The packets do not go through the host stack at all and are directly copied to the guest's memory. The interrupts may still occur (due to the reasons mentioned in part b), so interrpt overhead is NOT avoided by using SRIOV, and only packet copy overhead (across host and guest stacks) is avoided.

(d) Describe one optimization (other than SR-IOV) that can be used to avoid the overhead of one VM exit for every I/O request that must be submitted to the device. (You must describe the mechanism briefly, not just give a name.)

Ans: By using virtio drivers, I/O requests can be batched before switching to root mode. These requests can be written in virtIO queues or in the shared memory (memory mapped I/O). The cost of one VM exit is amortized over the batch of I/O requests.

25. Consider the iterative pre-copy and post-copy based migration schemes discussed in class, with Xen-based implementations.

(a) Xen directly uses page tables of guests instead of shadow page table during regular operation, but shifts to using shadow page tables during migration. Explain why this shift is needed, and why shadow page tables are required during migration.

Ans: In iterative pre-copy, the VMM needs to know which pages are dirtied in each pre-copy round. This is done by maintaining shadow page tables. In each round, the shadow

page table is reinitialized. Every page accessed by the guest is marked as read-only by default in the shadow page table. When a guest writes to a page, the page is marked as dirty, and the shadow page table entry is marked as writable. Therefore, the shadow page table is needed to keep track of dirty pages. Making such changes directly in the guest page table will lead to incorrect entries in the guest page table.

Shadow page tables are also used in post copy. The shadow page table entries indicate which pages are not yet present in the target machine. Upon a trap, the corresponding page is fetched from the source over the network. Once again, this information cannot be populated into the original guest page table, so a shadow is used.

(b) Consider a page table entry (PTE) in a guest page table that has read/write permissions for userspace, but this PTE is not part of the shadow page table yet. Describe what happens when the guest user reads this page and a page fault occurs. Clearly describe what entries with what permissions are added to which page tables.

Ans: A new entry is added to the shadow page table mapping from GVA to HPA. PTE is marked read-only (this is important, we do not give write permission yet.)

(c) Repeat the above question in the case when the guest performs a write on a page, and the PTE is not part of the shadow page table yet.

Ans: Shadow page table entry is added if not already present. It is also given read/write permissions. (The page is now part of dirty bitmap, so write permission is given, and future writes for this page in this round do not lead to a trap.)

(d) Mention one performance metric that is expected to be better in the case of post-copy migration as compared to iterative pre-copy migration, and explain your answer.

Ans: Total migration time will be lesser because pages are transferred only once across the network (pre-copy might transfer a page multiple times across multiple rounds). Number of pages transferred is also fewer.

Multiple Choice Questions

1. Consider the design of a demand-filled look-aside cache in the Memcache paper studied in class. Suppose a client performs a get on a key in the cache and there is a cache miss. Which of the following statements best describes what happens next?

   (a) The cache fetches the key-value pair from the backend and returns it to the client. The cache also stores a copy of the key-value pair.

   (b) The client fetches the key-value pair from the backend. The client also puts a copy of the key-value pair in the cache.

   (c) The client fetches the key-value pair from the backend, but does not update the cache.

   (d) The cache fetches the key-value pair from the backend and returns it to the client. The cache stores a copy of the key-value pair only if the key is popular historically, and discards it immediately otherwise.

   Ans: b

2. Consider the design of a demand-filled look-aside cache in the Memcache paper studied in class. Suppose a client wishes to put/write a key-value pair. Which of the following statements best describes how a put operation is performed?

   (a) The client puts the key-value pair in the cache, and the cache updates the backend synchronously.

   (b) The client puts the key-value pair in the cache, and the cache updates the backend asynchronously.

   (c) The client puts the key-value pair in the backend directly, and does not update the cache in any way.

   (d) The client puts the key-value pair in the backend directly, and invalidates the old value in the cache.

   Ans: d

3. Consider the concept of leases discussed in the Memcache paper studied in class. Which of the following statements is/are valid scenarios where a lease is used?

   (a) The lease is issued by the cache on a get request, and must be presented by the client on a subsequent put request.

   (b) The lease is issued by the backend on a get request, and must be presented by the client on a subsequent put request to the cache.

   (c) The lease is issued by the cache on a get request, and must be presented by the client on a subsequent put request to the backend.

   (d) The lease is issued by the backend on a put request, and must be presented by the client on a subsequent get request to the backend.

Ans: b

4. Consider the in-memory index maintained in the Store servers in the Haystack paper studied in class. The index maintains a mapping from the photo key/alternate key to which important piece of information about the photo?

   (a) Offset of the photo on disk

   (b) Inode number of the photo

   (c) Open file description of the photo

   (d) Logical volume number of the photo

   Ans: a

5. Consider the in-memory index maintained in the Store servers in the Haystack paper studied in class. When a store machine restarts after a crash, what is the status of the in-memory index? Select all that apply.

   (a) The in-memory index must be constructed from scratch by reading all the information on disk.

   (b) The in-memory index can be initialized from a checkpointed version on disk. This checkpointed index is always up to date after a crash, and can be used right away after a restart.

   (c) The in-memory index can be initialized from a checkpointed version on disk. However, the index checkpointed on disk may not have the latest updates made on disk, and must be updated to reflect the latest changes.

   (d) The in-memory index cannot correctly reflect the information about deleted photos after a crash.

   Ans: c

6. Consider the following alternate design to Haystack for storing photos on a server: each photo is stored in a separate file in the top-level root directory, using a regular Linux filesystem. How does this alternate new design of photo storage compare with the Haystack design studied in class with respect to the number of disk accesses required to read a photo, given the name/key of the photo/file? Assume that the root inode and all its data blocks are always in memory. Assume that the disk buffer cache is limited and can accommodate some but not all photos in memory. Select all options that are true.

   (a) It is possible to access some popular photos with no disk access in the new scheme.

   (b) Every photo read will require at least three disk accesses in the new scheme.

   (c) The new scheme will require more disk accesses than Haystack for the unpopular photos.

   (d) The new scheme will result in the same number of disk accesses as with Haystack for all photos.

   Ans: a, c

7. Consider the Bigtable paper discussed in class. Suppose a client C1 is accessing some table rows stored at tablet server S1. C1 has cached the metadata/location information of S1, and continues to read the table from S1 without looking up the location information in the metadata tables every time. After some time, server S1 loses its connection to Chubby, with the result that the master reassigns the tablets of S1 to another tablet server S2. A new client C2 looks up the metadata information in Chubby and starts accessing the tablets stored at S2, and even modifying them. At this point, if client C1 continues to read the old version of the table from S1, it may potentially access stale information. Which of the following mechanisms does Bigtable have in order to prevent this problem? (Note that the specific mechanism to address this problem has not been discussed in class, and you are not expected to recall the answer. But you should be able to reason about it from the basic concepts discussed in class, and arrive at an answer that makes sense.)

   (a) A tablet server that loses its connection to Chubby should stop serving clients.

   (b) A master should never reassign tablets of a server that has lost its connection to Chubby.

   (c) Tablet server S2 should update all previous tablet servers like S1 that held the tablets in the past.

   (d) Bigtable has no mechanism to handle this scenario, and applications will have to handle temporary stale values.

   Ans: a

8. Bigtable stores persistent state at the tablet servers in SSTables on GFS. List all the benefits that result from this design decision.

   (a) Because SSTables are immutable, they can be accessed without locking, easing concurrency control.

   (b) It is enough to store tablets at one server alone, since GFS takes care of replication.

   (c) The efficient file format of SSTables allows all information to be stored in memory for quick access.

   (d) Because SSTables are lexocographically sorted, a merged view across multiple SSTables is easy to obtain during a read operation.

   Ans: a, b, d

9. Consider the Bigtable paper studied in class. Recall that the information about which tablet servers store which tablets is maintained in a 2-level hierarchy by Bigtable in the following manner. The root tablet published in Chubby contains the locations of all metadata tablets, and each metadata tablet in turn contains information about the the range of rows in a tablet and the location of the tablet server serving that tablet. Assume that each location information entry is 2 KB in size and the size of a tablet is 512 MB. What is the maximum size of a table (in units of number of tablets) that can be stored in the system such that all location information can fit in this 2-level hierarchy correctly? Assume 1K = 2^{10}.

(a) 2^{36}

(b) 2^{64}

(c) 2^{34}

(d) 2^{17}

Ans: a

10. The Dynamo system studied in class uses the concept of virtual nodes when performing consistent hashing. Which of the following statements is/are true about the advantages of using virtual nodes?

    (a) Virtual nodes allow a node to be responsible for multiple key ranges in the keyspace, thereby ensuring better load balancing.

    (b) When a node leaves or joins the system, virtual nodes ensure that the changes in load are more evenly distributed across the other nodes.

    (c) Virtual nodes allow a node replicate its key-value database across multiple virtual machines in the cloud for fault tolerance.

    (d) Virtual nodes allow a key to be replicated at multiple nodes having the same identifier in the keyspace.

    Ans: a, b

11. Consider the idea of vector clocks discussed in the Dynamo paper in class. Three nodes A, B, C in a distributed key-value store use vector clocks to track the versions of the key-value pairs stored in the system. A client stores various versions of the value corresponding to a key at various nodes in the following manner. Suppose the client stores a value V1 at node A with an initial vector clock of [(A,1), (B,0), (C,0)]. The client then reads the value V1 from A and writes another value V2 to B. Then, two other clients read V2 from B, and write updated versions V3 and V4 at A and C respectively. Another client then reads the values V3 and V4, reconciles any conflicts, and writes an updated merged value V5 at node A. What is the vector clock of this value V5 stored at A?

    (a) (A,3), (B,1), (C,1)

    (b) (A,2), (B,2), (C,1)

    (c) (A,2), (B,1), (C,1)

    (d) (A,3), (B,1), (C,2)

    Ans: a

12. Which of the following is/are the key differences between the Memcache and Dyanmo key-value stores?

    (a) Dynamo uses a shared-nothing architecture for scalability while Memcache does not.

(b) Memcache is primarily useful for serving read-intensive traffic, while Dynamo can handle both read-intensive and write-intensive workloads.

(c) Dyanmo tries to store key-value pairs durably while Memcache does not.

(d) Memcache stores key-value pairs on disk while Dynamo is an in-memory key-value store.

Ans: b, c

13. Which of the following statements is/are true about the clone() system call with respect to namespaces?

(a) The clone system call creates a new namespace and places the calling process in it (except in the case of PID namespace, in which case the namespace change happens with the next child process).

(b) The clone system call creates a new namespace and places the calling process in it (for all types of namespaces, including PID namespace).

(c) The clone system call creates a child process and places this child in its own new namespaces (as per the flags provided as arguments).

(d) The clone system call does not change the namespace of the calling process.

Ans: c

14. Which of the following statements is/are true about the unshare() system call with respect to namespaces?

(a) The unshare system call creates a new namespace and places the calling process in it (except in the case of PID namespace, in which case the namespace change happens with the next child process).

(b) The unshare system call creates a new namespace and places the calling process in it (for all types of namespaces, including PID namespace).

(c) The unshare system call creates a new process that is placed in its own new namespaces (as per the flags provided as arguments).

(d) The unshare system call never changes the namespace of the calling process.

Ans: a

15. Which of the following statements is/are true about the setns() system call with respect to namespaces?

(a) The setns system call creates a new namespace and places the calling process in it.

(b) The setns system call is used to let the calling process join an existing namespace (for all namespaces, including PID namespaces).

(c) The setns system call is used to let the calling process join an existing namespace (except in the case of joining a PID namespace, in which case the change in namespace will happen for subsequent child processes).

(d) The setns system call never changes the namespace of the calling process.

Ans: c

16. Consider a process P1 that is the first process in a newly created PID namespace. P1 spawns a child P2, and the parent of P1 is P0. Another process Q1 joins this new PID namespace of P1, and Q1 spawns a child Q2. Which of the following statements is/are true?

    (a) Any orphan processes in the new PID namespace will be reaped by P1.

    (b) When process Q1 dies, it will always be reaped by P1.

    (c) Any orphan processes in the new PID namespace will be reaped by P0.

    (d) When process Q2 dies, it will always be reaped by P1.

    Ans: a

17. Consider a process P1 that has a PID=N1 in its namespace. Which of the following statements is/are true?

    (a) The process P1 is always visible with the same PID N1 in its parent's namespace.

    (b) The process P1 is always visible with the same PID N1 in its descendent namespaces.

    (c) No other process will have the same PID N1 in the namespace of P1, as long as P1 is active.

    (d) No other process will have the same PID N1 in any of the namespaces that P1 is visible in, as long as P1 is active.

    Ans: c

18. Consider two containers C1 and C2 located on a physical host running the Linux operating system. Which of the following is/are necessarily the same across C1 and C2?

    (a) The system call interface available to user applications and libraries

    (b) The operating system binary/image on which the containers are run

    (c) The system binaries for commands like ls, ps

    (d) Programming language shared libraries

    Ans: a, b

19. Which of the following statements is/are true about the Remus system for VM checkpointing studied in class?

    (a) The network output for an epoch is released from the primary VM after the backup VM acknowledges the checkpoint.

    (b) The network output for an epoch is released by the backup VM after the checkpoint is received from the primary.

(c) The execution at the primary VM continues speculatively into the next epoch after the checkpoint for the previous epoch is sent to the backup VM.

(d) The execution at the primary VM continues speculatively into the next epoch after the checkpoint for the previous epoch has been acknowledged by the backup VM.

Ans: a, c

20. Consider a key-value server running on a VM, which is using Remus to provide high availability to its clients. Which of the following statements is/are true?

(a) All get/put requests received from clients are processed both at the primary and backup VMs.

(b) All get/put requests received from clients are processed only at the primary VM, and the state changes are checkpointed at the backup VM.

(c) The key-value server application is required to run a distributed consensus protocol with its replica on the backup VM to maintain consistency of the checkpointed state.

(d) The key-value server application code need not be aware of, or explicitly communicate with, its replica on the backup VM.

Ans: b, d

21. Consider the Snowflock paper studied in class. In this system, the memory of a parent VM is copied on demand to multiple clones, in order to provide the VM fork abstraction. The memory of the parent is copied on write to avoid an initial copying overhead, and shadow page tables of Xen are used to implement this copy-on-write mechanism. Which of the following statements is/are true about the copy-on-write mechanism at the parent VM?

(a) The shadow page table entries always mark all pages in the parent VM's memory as read-only.

(b) The shadow page table marks writeable parent VM pages as read-only until the first valid write to a page happens.

(c) A copy of a memory page of the parent memory is made when a page fault occurs due to the parent trying to read the page for the first time after VM fork.

(d) A copy of a memory page of the parent memory is made when a page fault occurs due to the parent trying to write to the page for the first time after VM fork.

Ans: b, d

22. An application server running on a VM is exploring two different techniques for scaling up to multiple replicas on demand. In the first technique, the application server has several VMs booted up and ready for use. Upon detecting overload, the application server copies its server image and state to the extra VMs that are required to handle overload, and starts running from multiple replicas. In the second technique, extra VMs are not spawned beforehand. Instead, the required VMs are spawned and application server image/state copied only after detecting

overload. In both cases, the entire application code is copied to the new replicas one by one, and not via multicast. How do these techniques compare with achieving scale up using Snowflock?

(a) The first technqiue is expected to cost more (in terms of VM infrastructure cost) as compared to Snowflock.

(b) The first technique is expected to take lesser time to start the application on the new replicas as compared to Snowflock.

(c) The second technqiue is expected to cost more (in terms of VM infrastructure cost) as compared to Snowflock.

(d) The second technique is expected to take lesser time to start the application on the new replicas as compared to Snowflock.

Ans: a

23. Suppose a VM running a web server needs to be migrated live, without disrupting any of its active client connections. The migration techniques take down the network interface at the source host at the beginning of the stop-and-copy phase, restart it at the target host. Which of the following is/are possible ways of how the network interface on the VM can be configured during this migration process? Assume regular TCP/IP is used at the server.

(a) The network interface at the target host can have a different IP address from the source host, but both addresses must belong to the same IP subnet and must be announced via ARP.

(b) The network interface at the target host can have a different IP address from the source host, and both addresses need not belong to the same subnet.

(c) The network interface at the target host must have the same IP address as that at the source host, but can have different MAC addresses. The new IP to MAC mapping will be advertised via ARP.

(d) The network interface at the target host must have the same IP address as well as MAC address as that at the source host. Live migration will not work correctly even if one of these addresses changes.

Ans: c

24. Which of the following statements is/are true about the various VM live migration techniques?

(a) A migration technique that iteratively pushes VM state to the target before doing stop-and-copy is expected to have a lesser overall migration time as compared to a technique that transfers all state during stop and copy alone.

(b) A migration technique that iteratively pushes VM state to the target before doing stop-and-copy is expected to have a lesser downtime during the stop-and-copy phase as compared to a technique that transfers all state during stop and copy alone.

(c) A migration technique that iteratively pushes VM state to the target before doing stop-and-copy may end up transferring more data over the network as compared to a technique that transfers all state during stop and copy alone.

(d) A migration technique that iteratively pushes a subset of VM state to the target before doing stop-and-copy is expected to have a lesser disruptive impact on application performance post migration as compared to a technique that pulls most VM state from the source on demand after stop and copy.

Ans: b, c, d

25. Which of the following statements is/are true about paravirtualiation in Xen as studied in class?

(a) When a trap occurs for a guest domain, a major part of the trap handling is performed by the guest interrupt handlers in the guest domain.

(b) When a trap occurs for a guest domain, all trap handling happens within the Type 1 Xen hypervisor itself.

(c) Guest interrupt handlers in Xen domains can invoke privileged operations when handling interrupts.

(d) Guest interrupt handlers in Xen domains cannot invoke privileged operations when handling interrupts.

Ans: a, d

26. Which of the following statements is/are true about paravirtualiation in Xen as studied in class?

(a) When a guest domain is running, the page table pointed to by the CR3 register resides in the guest OS memory.

(b) When a guest domain is running, the page table pointed to by the CR3 register resides in hypervisor memory.

(c) When a guest domain is running, the page table pointed to by the CR3 register is updated by the guest OS directly.

(d) When a guest domain is running, the page table pointed to by the CR3 register is updated by the Xen hypervisor.

Ans: a, d

27. Which of the following statements is/are true about IO virtualization by emulation in QEMU/KVM (including virtio optimizations) as studied in class?

(a) If an interrupt occurs when a guest VM is running in VMX mode, the VM always exits to root mode to let the host handle the interrupt.

(b) If an interrupt occurs when a guest VM is running in VMX mode, the VM need not exit to root mode, and can handle the interrupt itself.

(c) When a guest user process issues an I/O system call in VMX mode, the VM always exits to root mode for every I/O request.

(d) When a guest user process issues an I/O system call in VMX mode, the I/O operation is always emulated by the KVM kernel module in root mode.

Ans: a

28. Which of the following statements is/are true with respect to shadow page tables for memory virtualization?

(a) The VMM must always maintain one shadow page table for every active process within every guest VM.

(b) A guest OS must maintain one guest page table for every active process it is running.

(c) The VMM must maintain at least one shadow page table for the currently executing process within every guest VM.

(d) The VMM can simultaneously maintain multiple shadow page tables for the different active processes running within a guest VM.

Ans: b, c, d

29. Consider the uncooperative swapping technique for VM memory reclamation studied in class. Which of the following statements is/are true?

(a) When a VMM performs uncooperative swapping to reclaim memory, the guest VM will not notice any changes to its guest virtual to guest physical address mappings stored in guest page tables.

(b) When a VMM performs uncooperative swapping to reclaim memory, the page fault that occurs due to a reclaimed memory page being accessed will be serviced by the VMM and not by the guest OS.

(c) When freeing up memory via uncooperative swapping, the VMM can only free up those guest VM pages that are not allocated to any guest process within the guest VM.

(d) When a VMM performs uncooperative swapping to reclaim memory, the page fault that occurs due to a reclaimed memory page being accessed will be serviced by the guest OS and not by the VMM.

Ans: a, b

30. Consider the VMWare workstation VMM studied in class, that performs dynamic binary translation. Which of the following statements is/are true?

(a) The translator logic of the VMM typically runs with a higher privilege (i.e., in a ring with more privileges) than the translated guest OS code itself.

(b) The translator logic of the VMM always runs at the same privilege level as the translated code guest OS code.

(c) When a translated code block (called the compiled code fragment) finishes execution, the VMM translator logic is always invoked to translate the next basic block.

(d) The translation cache in the VMM memory contains only those parts of the guest OS code that consist of sensitive but unprivileged instructions.

Ans: a

31. Consider the hardware-assisted QEMU/KVM hypervisor design studied in class. Which of the following statements is/are true when a CPU is executing guest VM code in VMX mode?

(a) The host context to switch back to upon VM exit is mapped into the top 4MB of the guest address space.

(b) The host context to switch back to upon VM exit is available to the CPU as part of the VMCS structure.

(c) The VMM is not part of the address space of guest processes.

(d) The VMM occupies the top 4MB of the address space of guest processes.

Ans: b, c

32. Consider the definitions of sensitive and privileged instructions as discussed in class.

(a) The popf instruction (pops the top of the stack and sets the eflags CPU register) is a sensitive and privileged instruction.

(b) The popf instruction (pops the top of the stack and sets the eflags CPU register) is a sensitive but unprivileged instruction.

(c) The instruction that sets the CR3 register (pointer to the page table) is a sensitive and privileged instruction.

(d) The instruction that sets the CR3 register (pointer to the page table) is a sensitive but unprivileged instruction.

Ans: b, c