10 minutes

Score

6 out of 10

answers are hidden.

quiz: 6 out of 10
ir 14 at 6:44pm
took 10 minutes.

## Question 1

2 / 2 pts

1. Which of the following instructions would a hypervisor running on an Intel x86 CPU use to enter a virtual machine?

- VMLAUNCH
- VMSTART
- World Switch
- VMENTER

## Question 2

2 / 2 pts

2. Which of the following statements most accurately represents the treatment of the VMCS content?

- The hypervisor and processor both play roles in populating VMCS content
- Neither the hypervisor nor the processor are responsible for populating VMCS content
- The processor is solely responsible for populating VMCS content
- The hypervisor is solely responsible for populating VMCS content

0 / 2 pts

## Question 3

3. The CR3 control register stores the location of the page location (eg. page table structures) currently in use by the processor?

- True
- False

0 / 2 pts

## Question 5

5. Which answer best describes the responsibilities of the hypervisor author when using VMX controls?

- The hypervisor author is required to handle all exits requested by the selected controls.
- The hypervisor author must enable the VMX in IA-32 control, and additionally
- **The hypervisor author can select any set of controls desired, regardless of which physical CPU is being used.**
- The hypervisor author must enable all controls provided by the CPU in use.



Scanned by CamScanner

10 minutes

Score

6 out of 10

wers are hidden.

uiz: 6 out of 10
/ 14 at 6:44pm
look 10 minutes.

## Question 1

2 / 2 pts

1. Which of the following instructions would a hypervisor running on an Intel x86 CPU use to enter a virtual machine?

* VMLAUNCH

VMSTART

World Switch

VMENTER

## Question 2

2 / 2 pts

2. Which of the following statements most accurately represents the treatment of the VMCS content?

* The hypervisor and processor both play roles in populating VMCS content

Neither the hypervisor nor the processor are responsible for populating VMCS content

The processor is solely responsible for populating VMCS content

The hypervisor is solely responsible for populating VMCS content

## Question 3

0 / 2 pts

3. The CR3 control register stores the location of the page location (eg. page table structures) currently in use by the processor?

True

* False

Question 5

5. Which answer best describes the responsibilities of the hypervisor author when using VMX controls?

The hypervisor author is required to handle all exits requested by the selected controls

The hypervisor author must enable the "secondary controls" unconditionally

* The hypervisor author can select any set of controls desired, regardless of which physical CPU is being used

The hypervisor author must enable all controls provided by the CPU in use

The hypervisor author is required to handle all exits requested by the selected controls

**GUEST VM PAGE TABLE**

| VA | PA | Page Metadata Bits | | |
|---|---|---|---|---|
| | | P | A | D |
| 0x1000 | 0x12000 | P | X | |
| 0x2000 | 0x14000 | P | X | |
| 0x4000 | 0x13000 | P | X | X |
| 0x5000 | 0x9000 | P | X | X |
| 0x6000 | 0x6000 | P | X | X |
| 0x7000 | 0x7000 | P | X | |

**NESTED PAGE TABLE**

| GPA | HPA | Page Metadata Bits | | |
|---|---|---|---|---|
| | | P | A | D |
| 0x6000 | 0x40000 | P | X | X |
| 0x7000 | 0x4A000 | P | X | |
| 0x9000 | 0x44000 | P | X | X |
| 0x12000 | 0x45000 | P | X | |
| 0x13000 | 0x51000 | P | X | X |
| 0x14000 | 0x55000 | P | X | |

Question: Consider the following sequence of guest VM instructions:

```
149c <mystery>:
149c:       0f 32                                rdmsr
149e:       89 04 25 54 41 01 00                 mov    %eax,0x14154
14a5:       5a                                   pop    %rdx
14a6:       59                                   pop    %rcx
14a7:       0f 01 04 25 38 41 01 00              sgdtl  0x14138
14af:       0f 01 0c 25 28 41 01 00              sidtl  0x14128
14b7:       0f 00 04 25 48 41 01 00              sldtl  0x14148
14bf:       0f 00 0c 25 52 41 01 00              strl   0x14152
14c7:       0f 01 3c 25 00 10 02 00              invlpg 0x21000
14cf:       c3                                   retq
```

For each instruction, state if an exit is possible for any reason, and give the type of exit an reason if so. You may assume the following:

- The page containing the mystery function (eg, the page containing addresses 0x100 – 0x2000), is present in memory with RX permission (Read + Execute)
- The page 0x14000-0x15000 is present in memory with R/W permission
- The page 0x21000-0x22000 is marked as "not present" in the current page table.
- The mystery function is running in CPL0.
- The paging structures for the above tables are properly present in memory

| Instruction | Exit possible (Y/N) | If exit possible, which exit, and under what circumstances would exit occur |
|---|---|---|
| **149c** | Yes | This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit |
| **149e** | No | |
| **14a5** | Yes? | #PF: Page Fault Exception. Stack might not be present) /Accessing Memory |
| **14a6** | Yes? | #PF: Page Fault Exception. Stack might not be present) /Accessing Memory |
| **14a7** | Yes | instruction cause VM exits if the "descriptor-table exiting" VM-execution control is 1. |
| **14af** | Yes | instruction cause VM exits if the "descriptor-table exiting" VM-execution control is 1. |
| **14b7** | Yes | instruction cause VM exits if the "descriptor-table exiting" VM-execution control is 1 |

| | | |
|---|---|---|
| **14bf** | Yes | instruction cause VM exits if the "descriptor-table exiting" VM-execution control is 1. |
| **14c7** | Yes | The INVLPG instruction causes a VM exit if the "INVLPG exiting" VM-execution control is 1. |
| **14cf** | Yes | If return instruction pointer is not within the segment limit page fault exits. |

**Question:** Describe how live migration of a virtual machine between hosts works.

**Answer:** There are two ways through which live migration between hosts can work:

1) Managed Migration : - This type of migration is performed by daemon thread running in the management of the source and destination VM. It is responsible for creating a new VM on the destination machine and co-ordinates transfer of live system over the network.

- The control software performs copying rounds which runs the complete scan of VM's memory pages. Although all the pages are transferred in this the first round, in subsequent rounds it is restricted to pages that were previously dirtied. Page tables managed by guest OS are exposed to real physical address to fill TLB so they need not be mapped.

- Xen inserts shadow page table underneath the running OS in the dirtied log pages. And these shadow tables are populated on demand by translating sections of guest page table. As translation is simple for dirty logging, it initially reads only mappings and if the guest tries to modifies it the resulting page fault is trapped by Xen.

- When the bitmap is copied to the control software, the xen's bitmap is cleared and shadow page tables are destroyed. When it determines that pre-copy phase is no longer valid the OS sends a control message requesting to suspend itself. Xen informs the control software and the dirty bitmap is scanned one last time for remaining inconsistent memory pages.

- Once the final information is received at the destination, the VM state on the source machine is discarded. Control software on the destination machine scans the memory map and rewrites the guest page table to reflect the addresses of the memory pages that it has been allocated. Execution is then resumed by starting the new VM at old VM checkpoint.

- The OS then restarts its device drivers and updates its notion of wallclock time. As the transfer of pages is OS – agnostic any guest OS is supported which required by a small virtualized stub to handle resumption.

2) Self Migration: - Self migration places the majority of the implementation within the migration of OS. In this design no modifications are needed either to Xen or to the management software running on the source machine, although a migration stub must run on the destination machine to listen for incoming migration requests, create an appropriate empty VM, and receive the migrated system state.

Self migration is much more harder because the OS must continue to run in order to transfer its final state. So in order to overcome this difficulty, we logically check point the OS on entry to a final two-stage stop and copy phase. Here, the 1st stage disables all OS

activity except migration and then performs final scan of dirty bitmap, which then clears the appropriate bit of each page transferred. Finally all the pages still marked dirty are copied to a shadow buffer. The final stage then transfers the contents of the shadow buffer where page updates are ignored during transfer.

**Question:** Describe which operations need to be intercepted for application virtualization solutions, and how this interception is performed.

**Answer:** Following operations need to be intercepted for application virtualization solutions:

- Create File
- Open File
- Delete File
- Create User
- Delete User
- Query System Time
- Set System Time
- Open Network Connection
- Load Device Driver
- Create New Process
- List Files in Folder
- Reboot System
- Terminate System
- Set File Permissions
- Query File Attributes

There are two ways by which interception is performed:

1) First approach is system call interception where calls like (open, create, delete file) are hooked and intercepted before the underlying kernel actually process them.
   - So the system intercepts the call and examines the information for resources requested.
   - Then it either passes the request to the OS or changes the namespace/ID in order to "segregate the resource" before passing the new request to the underlying OS.

2) Second Approach is Process Injection-The whole process is explained as follows:
   - A master or launch process creates a child process in suspended state.
   - Before the process starts, the master process "reaches into" the child's memory space.
   - The parent process would then scan the child's memory space and look for the signature of routines that needs monitoring and finally rewrites those in place with call to its own engine.

CMPE283 – Virtualization
Sample Exam

Name:_____

**You may consult the Intel SDM reference guide during this exam, if needed.**

Select the best answer for each question numbered 1-5.

1. _____ Under what circumstances will a processor wake from a HLT instruction?

A. A non-masked interrupt occurs
B. A system call occurs
C. A process context switch occurs
D. Either B or C can cause a wake
E. None of A, B, or C

Updated Question: In a non-virtualized environment using an intel cpu newer than 80386, the "move to CR3" instruction flushes which of the following?

2. _____ The "move to CR3" instruction flushes which of the following?

A. All non-global translations from the processor's TLB
B. All translations from the processor's TLB
C. All non-global translations from all processors' TLBs
D. All translations from all processors' TLBs.
E. None of the above

3. _____ Which of the following operations does the system BIOS typically perform during startup?

A. Configuring paging
B. Determining process scheduling characteristics
C. Booting secondary processors
D. None of A, B, or C
E. Answers A, B, and C are all operations performed by the BIOS during startup

4. True _____ True or False? VMX root mode provides the virtualization systems architect with the capability to detect previously non trappable sensitive instructions?

5. False _____ True or False? Kernel area address translations present in a process' page directory (page tables) must be global translations?

Consider the following instructions.

```
(A)  SUB   %RAX,      $0x28      # subtract 0x28 (hex) from the value in the RAX register
(B)  MOV   %RAX,      %CR3       # move the content of the RAX register to the CR3 register
(C)  LGDT  *%RAX                 # load the GDT register with the content of the memory
                                   location pointed to by RAX
(D)  VMXON      *%RAX            # enable VMX operation on the current CPU
(E)  SGDT  *%RAX                 # store the content of the GDT register to the memory
                                   location pointed to by RAX
(F)  CPUID                       # gather CPU information
(G)  MOV   %RAX,      mem_loc    # move the content of the RAX register to mem_loc
```

6. Which instructions from the above set A-G, if any, may fault with a protection violation? Under what circumstances would each fault (if any)? in a non virtualized environment

B => if current privilege is greater than 0.
C => if current privilege is greater than 0.
  => memory location pointed is not accessible (read-only)
D => if current privilege is greater than 0.
  => memory location pointed is not accessible (read-only)
G => memory location pointed is not accessible (read-only)

Not in midterm

7. Which instructions from the above set A-G, if any, should a VMX-enabled VMM intercept while running guest VM code? Why?

C => yes if shadow paging, no if nested paging.
D => yes, must exit
E => yes, must exit
F => yes, must exit

8. Which instructions from the above set A-G are sensitive but not trappable instructions?

E

Consider again the following instruction:

```
MOV  %RAX,      mem_loc
       64 bits
```

You may assume the following:

- The page containing 'mem_loc' is regular memory and is marked 'present' in the current page table
- The page containing the MOV instruction and the page containing 'mem_loc' are defined as CPL=3
- No other privilege or page protection violations occur during the MOV operation
- Nothing is previously cached in memory
- This is the first time the page containing 'mem_loc' is being accessed
- All the paging and MMU structures are fully present and resident in memory
- The operation is occurring in a non-virtualized environment.

not in mid-term

9. What is the maximum number of memory accesses that might occur to complete the operation? Partial credit given for partially-correct results, provided an explanation is given.

18 → 20

**BONUS CHALLENGE QUESTION:**

10. In the scenario described in question 9, does your answer change if the assumption that the paging and MMU structures are fully present and resident in memory is removed? If so, why, and what is the new answer?

arbitarily many

*Mid term - 2*

*Sample mid-term. Its not a trick question.*

1. *Prior to Hardware assited virtualization – could you run a virtualization on x86*
    a. *False. It was done by VMware*
2. *A.*
3. *B Lazy switch – leave context switch ...*
4. *A. – virtual space is always 32 bits. For 64-bit – Not enough information provided.*
5. *C . 32-bit mde, the right ans is D*
6. *True.*
7. *True.*
8. *A. Intrupt Descipter table.*
9. *D*
10. *B. Store full state GSA and RSA*
11. *Give example. Enable "must enable the controls" eg CPUID. Compatible with all CPU , so use old controls*
    a. *Nested paging is advanced.*
12. *CPUID – Always going to exit. Hyperviser main loop. Emulate CPUID eg turn off thermal monitoring. Advanced RIP.*

---

*Mid term -3*

1. *VMM hide / mask CPU features '*

*5. A*

*6. True*

*7. D*

*9. A, D*

*11. Global Pages are used to map Kernal address base. Less overhead in flushing TBL.*

---

Sample exam

Sample Quiz

1,. A; 2. A; 3. D; 4. True; 5. False;

5. Global translations is not must.

6. Protection violation #GP.

Page fault, previledgs level zero

B, E,

Anytime memory is touched, Protection violation can occur

7. eg nested paging. DEF . VMXON and CPUID cannt be turned off
8.
9. 20+
10. Page fault. Unknown access

---

```
Populate VMCS(hs, gs, ctrls)
While(1) {
        Select VMCS/VCPU
        Vmlaunch/resume

       ? which exit
       switch(exit reason){
              case HLT:

              case CPUID:

       }

       .
       .
       .
       events – check for panding interrupt
}
```

Stop on exit i.e. ctnrls
Min # of controls – 6-7
Max # cntrl - 61
Average - 40

Overhead in RED. Therefore, the fastest hyperviser, reduce the number of exit and reduce the RED.

---

# 283: Week 6 Contd..

KVM is the hypervisor that is in Linux source code

Every VMM has a handler that handles the exits

Processor based controls control things inside the CPU

Pin based controls control things outside the process. Like I/O events, like interrupts

VMCs has a GSA and a HSA

1. Before a VM is started populate the VMCS. VMCS(hs, gs, controls)
2. While(1)
    1. (Select VMCS/VPCU) (based on scheduling algorithm)
    2. VM Launch / VM Resume. What process is chosen to run in the VM.
    3. An exit has stopped the process (Reduce the number of exits so that the HV can use time effectively. There is an overhead in all the time that we use to handle each kind of exit)
    4. Which Exit?
    5. Check the exit that stopped a process: Switch (exit reason)
        1. give each exit something to do
            1. kvm has about 40 diff types of exit
            2. example: handler_cpuid()
        2. events (emulation for all the interrupts). Event Injection
    6. Advance the RIP and perform the next instruction.

**How to channel the interrupts into the VM?**

We just use an emulated version of the device (example: sound card) in the VM.

Something like a virtual sound card. Look at the Guest VM memory, take it to the main memory, give it to the actual sound card, play the sound, give response to the guest VM.

**What happens if guest VM did While (1);**

You're screwed unless your interrupt exiting is turned on. Always, Always, Turn on Interrupt Exiting.

**What happens when there is an exit:**

10/18/2

Answered above in the flow.

**How many exits are there?**

Around 40.

**How many exits should you handle?**

Architecture says you must handle 6 or 7 exits

**What happens to RIP?**

Incremented. But in certain exits you do not increment the RIP.

**How to shut down VM?**

While ("live" cpus/vmcs) !shutdown)

**Note:** Send answer to professor. Free evaluation!