Virtualization of the physical memory and memory mapped IO

The concept of a Working Set

A virtual system can be configured to use RAM memory much larger than the host could allocate at any time. Also, allocating and locking too much of the host memory could cause trashing due to insufficient host working set during the virtual machine run.

We determine the optimal working set for each virtual machine based on the host available memory and guest total requirement. For example, if a host has 512Mb and a guest requires 256Mb, we can lock 64Mb at a time. **The exact calculation is still to be determined.** How much will system let us lock?

For the rest, we rely on the host paging

Virtualizing page tables

Guest OS keeps its own set of pages that it uses for paging. However, that set is not actually used; instead, it is being shadowed by VMX monitor's set of pages inaccessible to the guest software, and under the control of its paging framework. This text explains the methods to keep those two sets in sync. Besides correctly mapping memory pages, we need to take care of keeping real and guest pages' A (accessed) and D (dirty) bits in sync.

Any attempt by the guest to read CR3 (page directory) will cause VMX exit. We emulate the instruction (find out which register is the target of a read) and store the value which guest "thinks" it is using (the one it had set earlier on).

When a guest tries to write to CR3 (PD), VMX will exit (if the value is not one of the CR3 target values specified in the VMCS.) For now we will set the target count to 0, thus causing the unconditional exit.

Real (VMX's) set of pages will be allocated from the non-pagable pool. Since for each 4K of guest memory we need 4 bytes of one PT (page table), or one entry PTE (page table entry), we need 1 PT (4K) to map each Mb of guest physical memory. If a guest is configured to use 256Mb, we need to allocate 256 PT's times 4K = 1Mb + 1 page for PD. This assumes 4K pages.

To track paging, we use method called virtual TLB.

The monitor maintains paging hierarchy that effectively caches *translations derived from* the hierarchy maintained by the guest software. Guest can freely modify paging hierarchy. Monitor's ('real') paging may go out of sync with it. CPU also regularly writes to guest pages its Accessed and Dirty bits: when a page is read from, CPU will set the "A" bit in the corresponding active PTE and PD. When a page is written to, CPU will set "A+D" bits. Monitor needs to propagate these changes into the guest virtual PD and PTEs.

We start our shadow paging at the start of a guest run. Although a VM is in the real mode with no paging enabled, we track it the same way for simplicity reasons. Instead of guest OS using paging, our own VM monitor running in the guest space sets up simple paging.

We allocate and lock memory for PD and all PTs, fill in physical addresses of all PTs into the PD's entries (max 1024 of them).

Terminology:
"PD" = Page directory (4K table)
"PDE" = Page directory entry (8 bytes entry for a total of 1024 PDEs within one PD)
"PT" = Page table (4K table)

"PTE" = Page table entry (8 bytes entry for a total of 1024 PTEs within one PT)
"F" = Frame, 4K page
"P" = Present bit
"A" = Accessed bit
"D" = Dirty bit
"VMM" = Virtual Machine Monitor, our code running v86 guest and transition states
"WS" = Working Set of pages that are always locked in memory

Initial State

VMX has to set CR0.WP for the paging unit to intercept guest supervisor writing to user read-only pages.
CR0.PE=1 – Enable Protected Mode, bit [0]
CR0.NE=1 – Numeric Error, enable native numeric error, bit [5]
CR0.PG=1 – Enable paging, bit [31]
CR4.VMXE=1 – Enable VMX mode, bit [0]

In real mode paging is not used, yet we run VMM with simple 1:1 paging implicitly set by the VMM.

Optimizations:

- Use available bits [11:9] in VMX paging entries (both PD and PT) to store this information:
    - Bit [9] – when set in a PDE, the corresponding guest PDE is a 4Mb page PTE (we still use both: PDE->PTE, though)

VMX Paging Unit response to loading a new CR3

Any write to CR3 will flush the TLB and possibly change the page table. However, when the CR4.PGE (Page Global Enable, bit [4]) is set, all pages with a G bit are not flushed when CR3 is reloaded (only within PTE's of 4K pages and PDE's of 4Mb pages). We keep track of G bit with 4Mb PDE's only.

We allow guest OS to use that feature for performance reasons. In the same manner, CR4.PGE is enabled in VMX mode. (One way to completely flush the TLB is to reset the CR4.PGE bit to zero.)

TODO: Well, we really don't use G bit effectively with the virtual TLB. Should we even enable CR4.PGE? Are VMX Exits flushing TLB completely?

We keep the same page for PD:
    For each PDE (out of 1024) that is Present
        If the PDE is a 4Mb page PTE and Global bit is set
            Keep the entry as-is
        Else
            Free a page that is linked from this one (PT)
            Mark the PDE as not-Present (clear all other fields also)

This way the actual CR3 address always stays on the same PD.

VMX Paging Unit response to INVLPG instruction

Get the effective guest virtual address from the instruction operand
If the PDE is a 4Mb page PTE
    Clear the PDE Present bit, making the frame it links to not-Present (PT)
    Free a page that is linked from this one (PT)
Otherwise clear the linked PTE Present bit, making the frame it links to not-Present

This instruction ignores the Global bit in page tables.

VMX Paging Unit response to Page Faults (#PF)

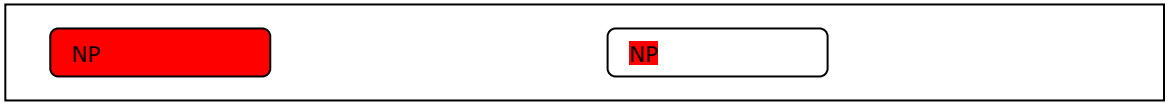
The #PF exception pushes the error code on the stack. We examine that code for the cause of a fault:

[0] P	Page was not present
[1] W/R	Faulting was a write (1) or a read (0)
[2] U/S	Code was in user (1) or supervisor (0) mode
[3] RSVD	Code trying to access a page whose PDE or PTE's reserved bit was not "0" (we don't use it)

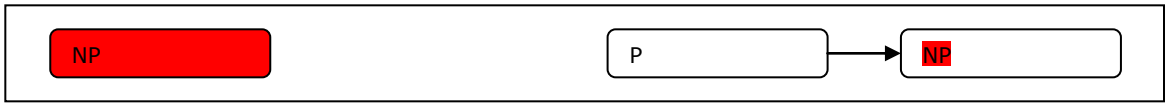
The #PF exception also stores the linear address that generated a fault in CR2 register.
The CS:EIP points to the instruction that generated the fault (unless it was a task switch)

First check simple cases where the guest would have caused a page fault.

If the fault was caused by a non-Present page (error code[0]=1)
If the PDE that faulted not-Present
If the corresponding guest PDE not-Present:

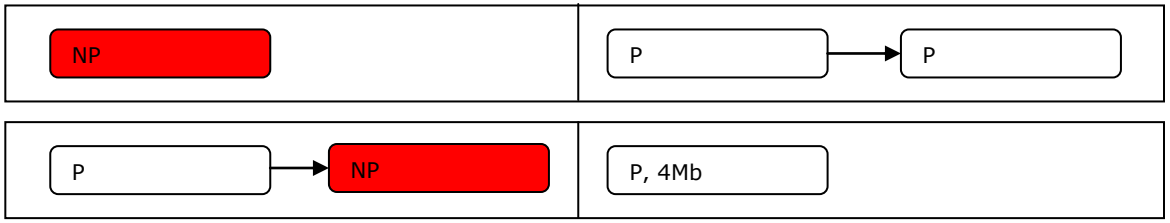


Or if the guest PDE is Present, but the guest PTE is not-Present:



Guest would cause #PF, send a #PF to the guest OS. Done.

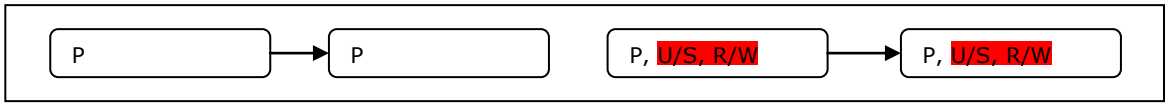
If both corresponding guest PDE and PTE are Present, but PDE or PTE are not



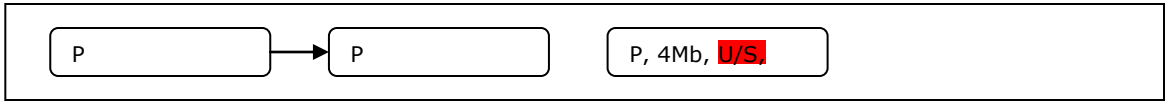
Goto Step 4

If the fault was not caused by a non-Present page (error code[0]=0). Page was present, but we had a permission fault. This can be a legitimate permission violation (if the active and guest attributes match) or our update to the Dirty bit on write (where we set up a read-only bit). In the former case we don't have to set Accessed and Dirty bits since effectively memory accesses really don't complete.

1. Guest PDE is a pointer to guest PTE



2. Guest PDE is a 4Mb PTE



Check the guest U/S and R/W against the faulting code access:

EC[2]:U/S	EC[1]:R/W	CR0:WP	P[2]:U/S	P[1]:R/W	Effect
S (0)	R (0)	0	0	0	
S	R	0	0	1	
S	R	0	1	0	
S	R	0	1	1	
S	R	1	0	0	
S	R	1	0	1	
S	R	1	1	0	
S	R	1	1	1	
S	W (1)	0	0	0	Supervisor writes to a S., read-only page
S	W	0	0	1	
S	W	0	1	0	
S	W	0	1	1	
S	W	1	0	0	Supervisor writes to a S., read-only page
S	W	1	0	1	
S	W	1	1	0	S. writes to a read-only user page, WP set
S	W	1	1	1	
U (1)	R	0	0	0	User accesses supervisor page
U	R	0	0	1	User accesses supervisor page
U	R	0	1	0	
U	R	0	1	1	
U	R	1	0	0	User accesses supervisor page
U	R	1	0	1	User accesses supervisor page
U	R	1	1	0	
U	R	1	1	1	
U	W	0	0	0	User accesses supervisor page
U	W	0	0	1	User accesses supervisor page
U	W	0	1	0	User writes to a user, read-only page
U	W	0	1	1	
U	W	1	0	0	User accesses supervisor page
U	W	1	0	1	User accesses supervisor page
U	W	1	1	0	User writes to a user, read-only page
U	W	1	1	1	

Error code[1] = R/W: (0=read, 1=write)
Error code[2] = U/S: (0=supervisor, 1=user)
PTE[1] = R/W: (0=read only, 1=writable)
PTE[2] = U/S: (0=supervisor access only, 1=user access)
CR0.WP: 1=fault on supervisor writes to user read-only pages

If a guest PDE is not a 4G PTE, calculate the effective combined permission:
PDE and PTE U/S effective permission is **P[2] = PDE[2] & PTE[2]**, if any one is marked as supervisor access only, the effective is a supervisor access.
The same is with read/write access: **P[1] = PDE[1] & PTE[1]**, if any level is marked as read-only, the effective is read-only.

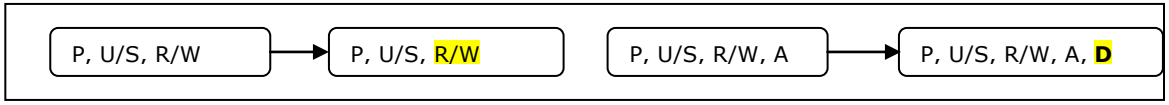
Fault if user accesses any page marked as supervisor: **F1 = EC[2]==1 && P[2]==0**
Fault if user writes to a read-only page: **F2 = EC[1]==1 && EC[2]==1 && P[1]==0**
Fault if CR0.WP is set and a supervisor writes to a read-only user page: **F3 = CR0.WP && EC[1]==1 && EC[2]==0 && P[1]==0 && P[2]==1**
Fault if supervisor writes to a supervisor read-only page: **F4 = EC[1]==1 && EC[2]==0 && P[1]==0 && P[1]==0**

If the guest would cause #PF (F1 | F2 | F3 | F4), send a #PF to the guest OS. Done.

At this point we know that the guest page state would not cause fault, so the fault originates from our page state for the purpose of virtual TLB management – update Dirty bit in this case using a read-only attribute.

If the guest PDE contains a pointer to the PTE for 4K pages (PS=0)

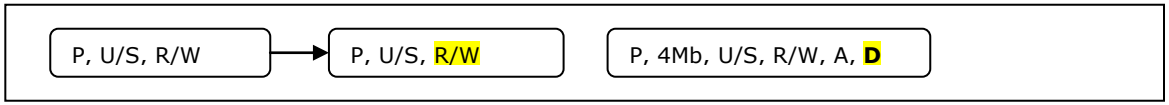
Set the D bit in the guest PTE
Copy the R/W bit from the guest PTE into the active PTE



Re-execute the failing instruction. Done.

If the guest PDE is a 4Mb PTE (PS=1)

Set the D bit in the guest PDE
Copy the R/W bit from the guest PDE into the active PTE



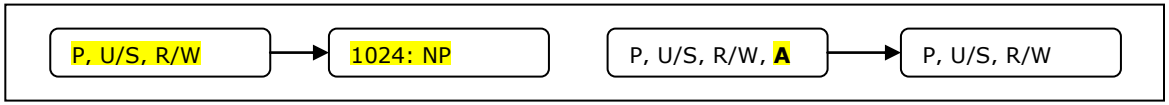
Re-execute the failing instruction. Done.

Another case to consider: If a guest PDE contains a guest physical frame address that is outside bounds of the allocated RAM memory, we delegate this address to the PCI subsystem where it gets compared to a set of PCI devices' MMIO ranges. Then, the access will either be emulated and routed to a device owning a MMIO range. If the address is not owned by any device, we will ignore the write and simply return 0xff's for reads.

Step 4: PDE is not-Present, guest PDE is Present - This is a case of virtual TLB fill. Set the active tables to correspond to guest.

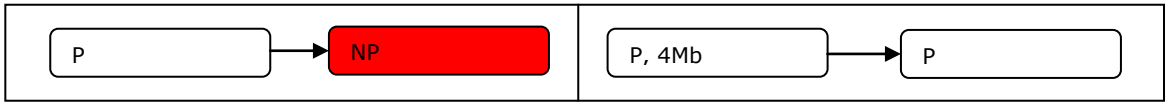
If the guest PDE contains a pointer to the PTE for 4K pages (PS=0)

Allocate a new 4K PT page
Clear new page to not-Present (all 1024 entries)
Link the new PT from the active PDE by its real physical frame address
Set the U/S, R/W bits in PDE identical to the guest PDE; also set PWT, PCD and G bits according to the VMX policy (TODO)
Set A in the guest PDE

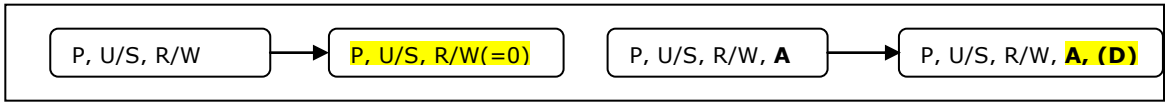


Continue into the next case below…

If PDE was present, but PTE was not-Present



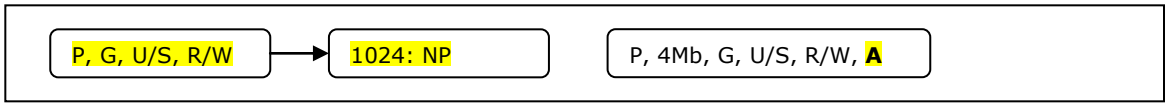
For a corresponding PTE (single entry):
Link the PTE to the physical frame of the corresponding guest memory page translated from the guest PTE (map it)
Set the U/S, R/W bits in PDE identical to the guest PDE; also set PWT, PCD and G bits according to the VMX policy (TODO)
Set A in the guest PTE
If the guest PTE has D bit still clear
    If the access was a write (error code[1]=1), set D bit in the guest PTE
    If the access was a read (error code[1]=0), clear R/W bit in the PTE (arm for read-only) so we get chance to set D bit when a write occurs.



Re-execute the failing instruction. Done.

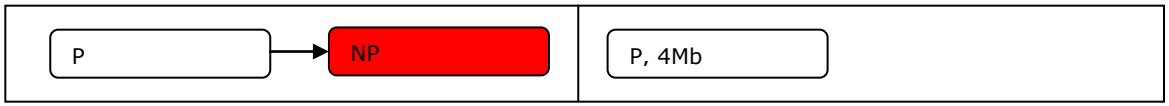
If the guest PDE is a 4Mb PTE (PS=1)

Allocate a new 4K PT page
Clear new page to not-Present (all 1024 entries)
Link the new PT from the active PDE by its real physical frame address
Set the G, U/S, R/W bits in PDE identical to the guest PDE; also set PWT, PCD bits according to the VMX policy (TODO)
Set A in the guest PDE

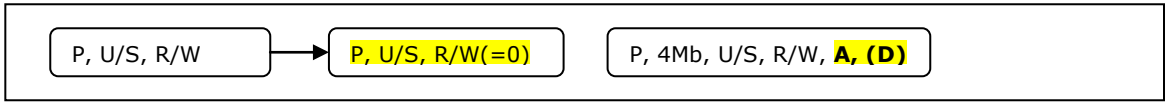


Note that G bit in PDE is not effective (it is ignored), but we use it to signal the TLB flushing logic not to flush this entry.
Continue into the next case below…

If PDE was present, but PTE was not-Present



For a corresponding PTE (single entry):
Link the PTE to the appropriate physical frame of the guest memory page translated from the guest PDE + offset within 4Mb
Set the U/S, R/W bits in PDE identical to the guest PDE; also set PWT, PCD and G bits according to the VMX policy (TODO)
If the guest PDE has D bit still clear
If the access was a write (error code[1]=1), set D bit in the guest PDE
If the access was a read (error code[1]=0), clear R/W bit in the PTE (arm for read-only) so we get chance to set D bit when a write occurs.



Re-execute the failing instruction. Done.