Wirtualizacja wspomagana sprzętowo - zalety, wady i zagrożenia

Julia Romanowska, Andrzej Pragacz, Marcin Pawłowski

27 listopada 2008

Spis treści

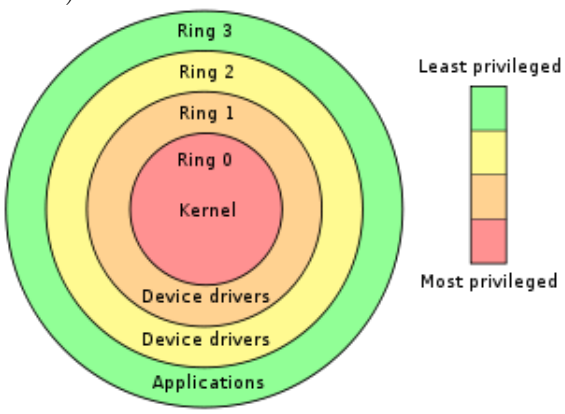
1 Wirtualizacja bez wsparcia sprzętowego

1.1 System ochrony pamięci x86

Kod systemu operacyjnego musi mieć prawo zrobić wszystko (bezpośredni dostęp do sprzętu), ale z drugiej strony zwykłe aplikacje nie powinny mieć możliwości spowodowania błędu krytycznego w systemie. Żeby to osiągnąć trzeba stworzyć jakiś rodzaj bariery sprzętowej. W x86 (procesorach i386 i lepszych) za izolację kodu uprzywilejowanego od kodu zwykłych programów odpowiadają poziomy dostępu:

- Każda strona w pamięci ma przypisany poziom dostępu – 2 bity, mówią one na jakim poziomie będzie wykonywać się kod w nich zapisany.
- Żeby przejść z jednego poziomu do drugiego trzeba przekroczyć specjalną furtkę – instrukcją procesora syscall/sysenter (wyjątek generowany przez program)

Wciąż obecny też jest stary (pochodzący jeszcze z okresu przed i386) mechanizm segmentacji pamięci za pomocą rejestru gs, który może posłużyć do izolacji kodu . Pamięć zostaje podzielona na segmenty. Każdy proces ma w rejestrze gs zapisany swój segment, a także jego długość. Zanim adres zostanie sprawdzony przez mechanizm stronicowania, następuje sprawdzenie czy adres nie wychodzi poza dozwolony obszar. Tylko proces o poziomie dostępu 0 może zmodyfikować swój rejestr gs (wskazujący na obecnie używany segment).



źródło: en.wikipedia.org [20]

1.2 Jak działa wirtualizacja

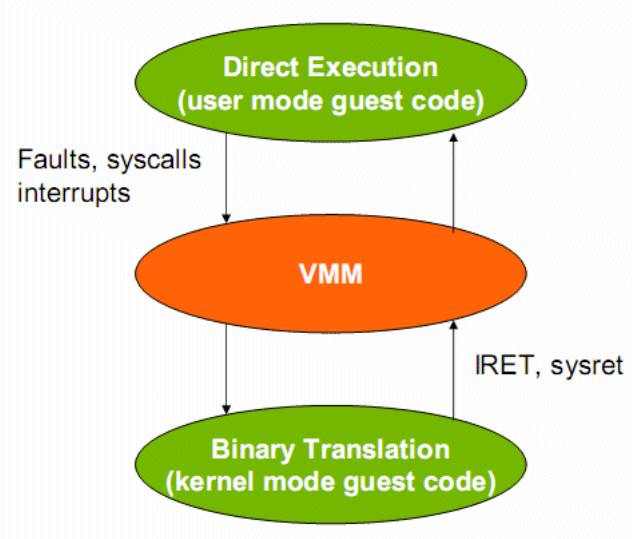
Trzeba stworzyć iluzję, że system operacyjny gościa ma pełne prawa - może wykonywać wszelkie instrukcje oraz ma dostęp do sprzętu. Na IBM s/370 i nowszych – procesor sam przechwytuje uprzywilejowane instrukcje wywołane przez gościa i generuje odpowiedni wyjątek przechwycony przez nadzorcę (VMM).

Procesory x86 nie były wyposażone w ten mechanizm. Posiadają 17 instrukcji uprzywilejowanych, których nie można przechwytywać, np. blokowanie przerwań. Ponadto żeby wirtualizacja była wierna gość nie może być w stanie stwierdzić, że pracuje na maszynie wirtualnej. Na x86 bardzo trudno go oszukać: może sprawdzić rejestr gs, którego dwa ostatnie bity to poziom dostępu – będzie miał coś innego niż 0 !! Co więcej, może chcieć zmienić swoją tablicę stron – to jest jego struktura danych i ma do niej prawa dostępu, więc żaden mechanizm mu tego nie zabroni. Co gorsza, tablice są obsługiwane przez procesor w sposób sprzętowy, więc nadzorca musi je zaktualizować, zanim gość użyje jakiegoś nowego adresu. Gość może też np. wywołać instrukcję popf (zmienia flagi ALU, ale także flagi systemowe, np. IF – blokowania przerywań) i teraz jako, że pracuje na wyższym poziomie niż 0, to procesor zwyczajnie zignoruje wszelkie zmiany, jakie chciał wprowadzić on we flagach systemowych. Trzeba więc zmieniać kod gość!! Niestety emulacja tradycyjna jest zdecydowanie zbyt wolna.

1.3 VmWare

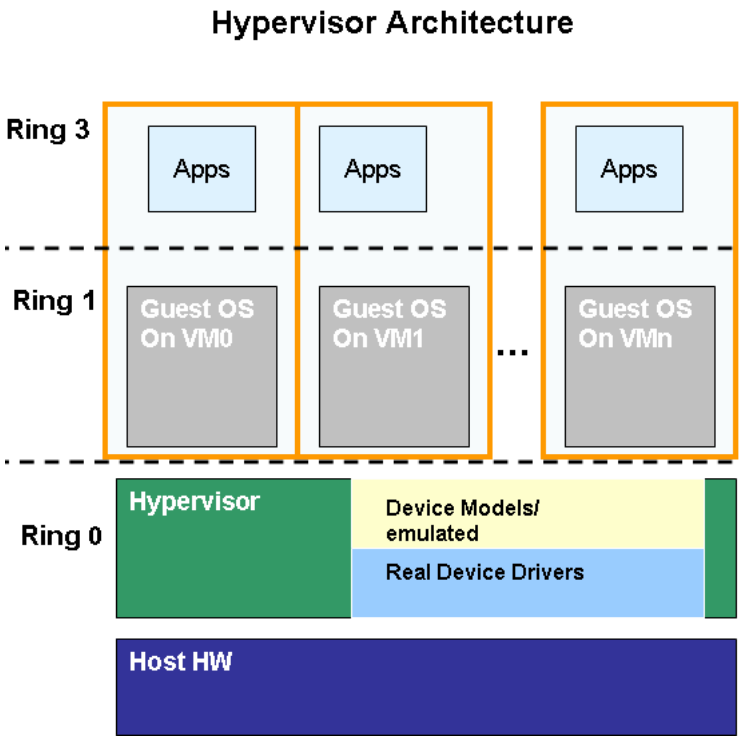
VmWare w ogóle nie zajmuje się kodem aplikacji działających w systemie gościa – są one wykonywane bezpośrednio na procesorze!! Dopiero, gdy wykonujemy kod systemowy na maszynie wirtualnej to następuje tłumaczenie binarne (Binary Translation):

- Kod systemu jest wejściem dla translatora, który przetwarza rozkazy uprzywilejowane na zestaw takich, które są bezpieczne. Po przetłumaczeniu kod trafia do cache (Translator Cache) – gdy mamy pętlę to tłumaczymy ją tylko raz, z czasem koszt translacji spada, bo cache staje się coraz bardziej efektywny.



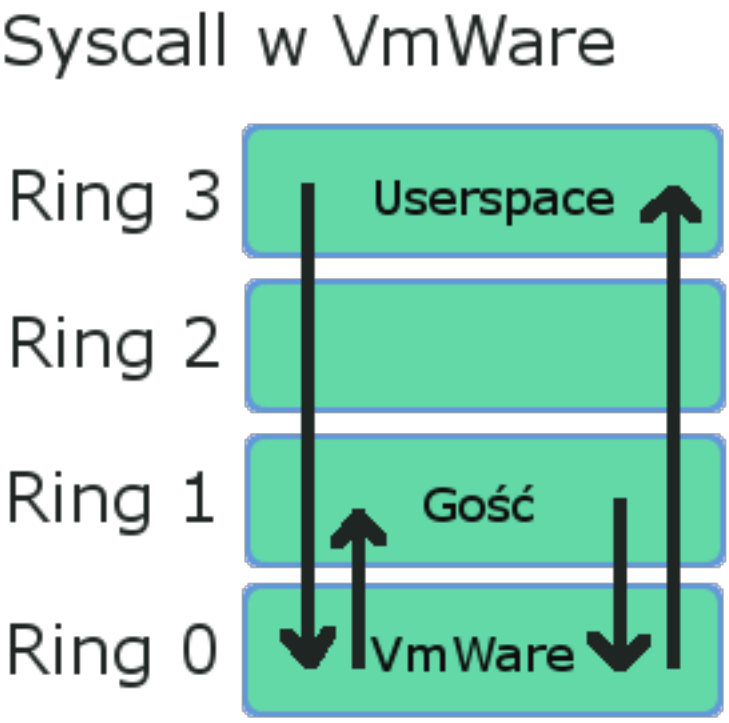
źródło: Hardware Virtualization: The Nuts And Bolts [21]

- Kod ulega wydłużeniu, trzeba więc śledzić jego przebieg, ponieważ odtąd skok o 100 bajtów może oznaczać, że trzeba przeskoczyć w kodzie przetłumaczonym o więcej – TC musi być bardzo inteligentne.
- Rozwiązanie to jest tańsze niż generowanie pułapek sprzętowych, takich jak w IBM s/370+ (choć producenci nowych procesorów bardzo intensywnie pracują nad wydajnością tych operacji i wkrótce może się to zmienić).
- Jeśli program z userspace gościa spowoduje wyjątek, VmWare go dostanie (bo pracuje na poziomie 0) i dopiero wtedy wykona odpowiedni fragment przetłumaczonego kodu systemu operacyjnego gościa.



źródło: Hardware Virtualization: The Nuts And Bolts [21]

Gość pracuje na poziomie 1 (jego przetłumaczony kod) i za pomocą segmentacji jest odizolowany od VmWare (dzięki temu unika się kosztownych TLB flushy przy przejściu od VMM do gościa). Po wykonaniu przetłumaczonego kodu, następuje sysexit – przejście do ring 0, dopiero wtedy VmWare przekazuje odpowiedź gościa programowi za pomocą prawdziwego sysexit (bo gość z poziomu 1 nie mógł od razu dojść do programu na poziomie 3). Stąd każde odwołanie do systemu gościa jest drogie: syscall natywne kosztuje 250-400 instrukcji, na VmWare ok. 3500 (dane dla Pentium 4 3.8GHz).



Operacje I/O oraz urządzenia:

- Trzeba emulować całe urządzenie – dostarczyć jego binarny interfejs (mamy tu podwójny koszt, gdyż nie dość, że zwirtualizowny system tłumaczy żądania przez sterowniki, to potem te żądania są znowu tłumaczone przez VmWare na odwołania do prawdziwego sprzętu).
- Trudno emulować nowoczesny sprzęt, wykonujący wiele pracy w hardware, poza tym producenci nie chwalą się wszystkimi rozwiązaniami które sprawiają, iż ich produkt jest wydajny!!

Zarządzanie pamięcią:

- Zwirtualizowny system operacyjny widzi pamięć logiczną nadzorcy jako fizyczny RAM – VMM musi tłumaczyć każdy adres drugi raz, używa do tego Shadow Page Tables - specjalne tablice stron, które mapują odwołania gościa do adresów fizycznych. Dzięki wykorzystaniu TLB w zdecydowanej większości przypadków nie ma żadnego narzutu, gdyż mapowanie jest w cachu procesora.

- Problem, pojawia się gdy zwirtualizowny system zmienia mapowanie pamięci – wtedy niezbędny jest udział VMM, który po takiej operacji tłumaczy wszystko, co zmodyfikował gość na odpowiednie wpisy w Shadow Page Tables (to kosztuje od 3 do 400! razy więcej niż w przypadku zmian w natywnej tablicy stron).

Ponadto VmWare ma kłopoty w translacji kodu jądra, który sam się modyfikuje (self-patching) – TC wtedy czasem zawodzi, na szczęście dla VmWare tego typu kod jest bardzo rzadko wykonywany w systemach operacyjnych (w Linuxie tej techniki używa np. SystemTap, by umożliwić obserwacje dowolnego fragmentu jądra, ale z drugiej strony nie powodować narzutu, gdy tego nie robimy).

1.4 Parawirtualizacja = Xen

Przy użyciu tej techniki trzeba zmodyfikować gościa (dla Linux wystarczyło około 2% kodu). Gość wie, że nie działa bezpośrednio na sprzęcie i „współpracuje" z VMM poprzez system hypercall. Zamiast odwoływać się do hardware lub wykonywać instrukcje uprzywilejowane wywołuje odpowiednią funkcję Xena:

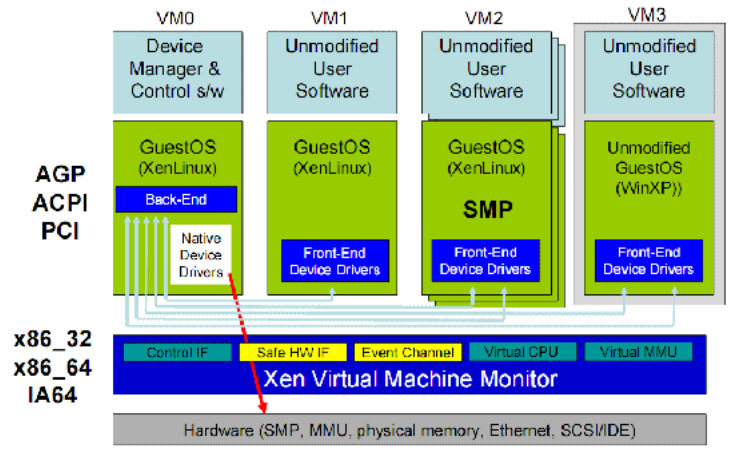
- Sam Xen jest wmapowany w przestrzeń adresową gościa tak by można było szybko do niego przejść.

- Xen udostępnia interfejs do uproszczonego I/O na wirtualnych urządzeniach oraz metody modyfikacji tablic stron - gość sam zwraca się do niego z prośbą o aktualizację swoich page table.

- Gość zdaje sobie sprawę, iż działa na maszynie wirtualnej i wie że operacje na tablicy stron są kosztowne. By zwiększyć wydajność może wstrzymywać zmiany w page tables, tak by wysyłać je w paczkach do Xen - jest to rozwiązanie o wiele szybsze.

- Gość może zarejestrować szybkie procedury obsługi wyjątków (zostaną one sprawdzone przez Xen, czy nie zawierają instrukcji krytycznych). Potem, gdy następuje wyjątek systemowy nie potrzeba żadnej zmiany kontekstu, te procedury są wywoływane na poziomie 0 - bo są bezpieczne!!

- Dzięki temu, iż gość wie o swojej sytuacji, dużo łatwiej przekazywać mu czas systemowy (lub go zmienić).

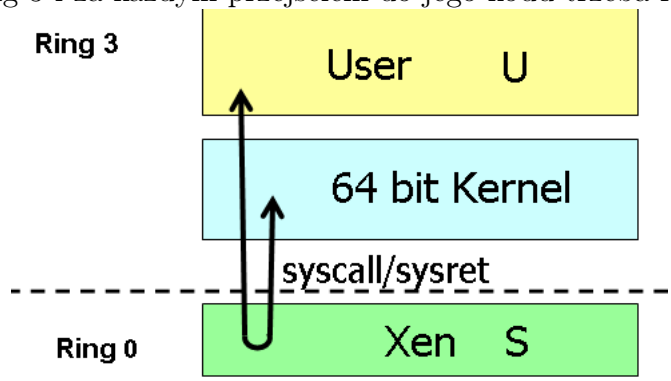


źródło: Hardware Virtualization: The Nuts And Bolts [21]

1.5 Koszt wirtualizacji

- Translacja binarna – kosztuje niewiele a z czasem prawie nic (dzięki TC).
- Syscall – koszt wysoki ze względu na liczne zmiany kontekstów.
- Wirtualne urządzenia – niska wydajność kart sieciowych i dysków (choć dzięki sprytnym sterownikom VmWare potrafi być dość szybki)
- Zmiany page table – bardzo drogie, aplikacje bardzo intensywnie korzystające z zarządzania pamięcią przez gościa mają spadki szybkości nawet do 20% natywnej.
- W przypadku Xen:
  - BT – brak.
  - Syscall – zykle dużo wydajniejsze niż przy VmWare, ale w niektórych przypadkach tak samo wolne.
  - I/O oraz urządzenia – koszt niewielki.
  - Zmiany tablic stron – dużo tańsze niż w VmWare, zwłaszcza gdy gość jest dobrze przygotowany do współpracy.

Xen traci na wydajności, gdy gość jest 64bitowy – musi on być wykonany w ring 3 i za każdym przejściem do jego kodu trzeba robić TLB flush!!



źródło: Hardware Virtualization: The Nuts And Bolts [21]

2 Pierwsze podejście do wirtualizacji wspomaganej sprzętowo - Ring -1, AMD-V i Intel-VT. Zasada działania tych rozszerzeń i problemy z nią związane. Wydajność w porównaniu do wirtualizacji realizowanej programowo za pomocą Ring 0-Ring 3.

2.1 Początki wirtualizacji sprzętowej

Historycznie najstarszą platformą wspierającą w sposób sprzętowy wirtualizację był System/370. Powstał w 1972 roku, dedykowany pod niego system operacyjny VM/370 był pierwszym obsługujacym sprzętową wirtualizację OS-em. VM/370 składał się z nadzorcy (Virtual Machine Monitora, VMM) VM-CP, na którym były uruchomione systemy takie jak CMS czy VM-CMS.

Działanie VMM było niezbyt wyrafinowane: Każda zarządzająca zasobami instrukcja uruchamiana w mniej uprzywilejowanym pierścieniu (czyli przez gościnny system operacyjny) powodowała tzw. pułapkę (Trap), przekazując sterowanie do VMM, który obsługiwał ją z uwzględnieniem żądań innych Guest OS-ów. Z czasem próbowano zmniejszać liczbę pułapek i redukować czas ich obsługi.

2.2 x86 a wirtualizacja sprzętowa

Wyprodukowany w 1986 roku procesor 80386 umożliwiał uruchamianie programów w tzw Virtual Real Mode (VM86). Był to specjalny tryb, pozwalający na uruchamianie w trybie chronionym aplikacji przeznaczonych pod tryb rzeczywisty (np. aplikacje DOSowe). Odpowiedzialność za obsługę przerwań spadała wówczas na system operacyjny, który musiał także emulować używane przez "wirtualizowaną" aplikację urządzenia.

Było to jednak za mało, żeby spełnić wymagania wirtualizacyjne określone przez Geralda Popka i Roberta Goldberga:

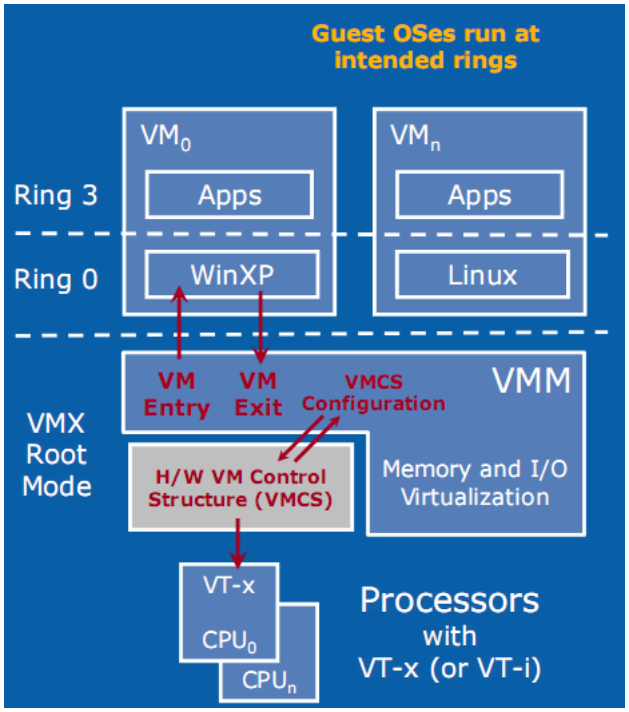
- Równoważność - program działając pod VMM powinien zachowywać się tak samo jak uruchomiony na fizycznej maszynie
- Bezpieczeństwo (Kontrola zasobów) - VMM musi mieć pod całkowitą kontrolą wirtualizowane zasoby
- Efektywność - Większość instrukcji musi być uruchamiana bez współudziału monitora wirtualnej maszyny

Zasadniczy problem polegał na tym, że architektura x86 nie poddawała się w pełni wirtualizacji. Nie wszystkie uprzywilejowane instrukcje mogły być "trapowane" tak jak na IBM S/370.Co ciekawe, architektury takie jak PowerPC czy Alpha ISA nie mają takich problemów i w pełni poddają się wirtualizacji.

2.3 Ring -1

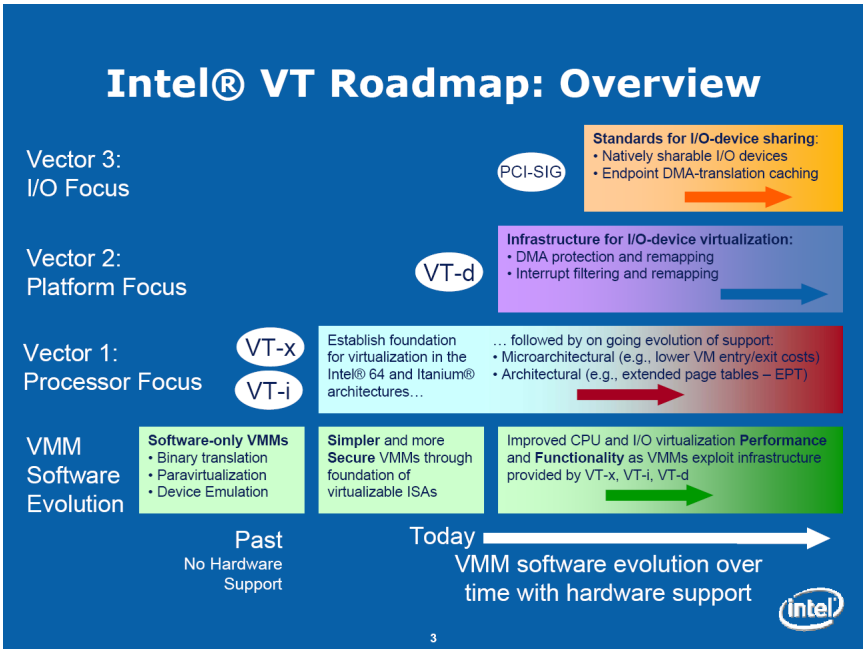
Problem z "niewirtualizowalną" architekturą x86 rozwiązano w sposób następujący: skoro niektóre instrukcje działają "niewłaściwie" w Ring 1, to pozwólmy im się wykonywać tam gdzie działają "dobrze", czyli w Ring 0. Stwarza to jednak oczywisty problem, bo VMM już nie jest w stanie w pełni panować nad tym, co robią wirtualizowane systemy operacyjne, gdyż współdzieli z nimi ten sam poziom uprzywilejowania. Podzielono więc najbardziej uprzywilejowany poziom na dwa:

- W pełni uprzywilejowany Ring 0 = Ring -1 (znany też jako VMX root)
- Mniej uprzywilejowany Ring 0 (znany też jako VMX non-root)



źródło: Hardware Virtualization: The Nuts And Bolts [21]

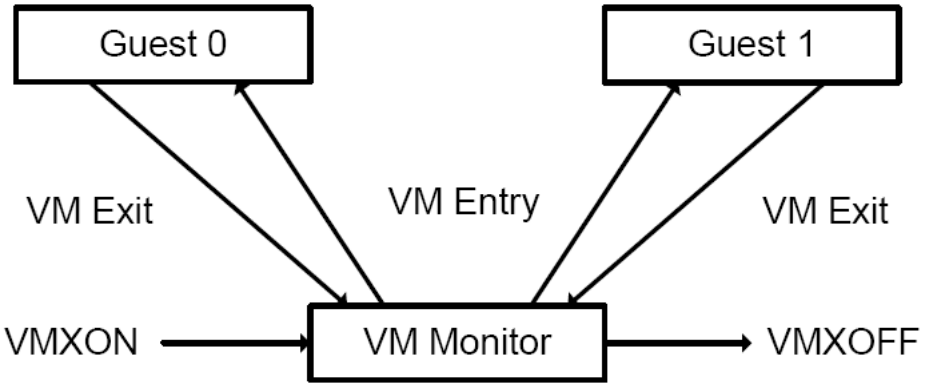
Należało także wprowadzić dodatkowy zestaw instrukcji pozwalający na przełączanie się między oboma trybami. Przodujący producenci procesorów tacy jak Intel i AMD, wprowadzili w 2006 roku na rynek odpowiednio technologie Intel-VT i AMD-V. Intel VT składała się początkowo z VT-x (technologia wirtualizacji dla IA-32) oraz VT-i (technologia wirtualizacji dla IA-64, czyli Itanium (nie mylić z x86_64)). Później została dodana VT-d (Virtualization Technology for Directed I/O) i VT-c (Virtualization Technology for Connectivity)



źródło: "Intel to introduce new virtualization capabilities with six-core CPU Nehalem" [22]

2.4 Intel VT-x

Intel VT-x to technologia wirtualizacji (Virtualization technology) przeznaczona dla procesorów IA-32. Wcześniej znana była pod nazwą "Vanderpool".



źródło: Intel Vanderpool technology for IA-32 Processors Preliminary specification [1]

2.4.1 VMCS

VMCS (Virtual Machine Control Structure) to strukura danych opisująca przejścia i wyjścia ze stanu VMX non-root. Opisuje m.in. przy jakich zdarzeniach ma nastąpić tzw. VM EXIT (wyjście z maszyny wirtualnej w celu przekazania sterowania do VMM) Procesor utożsamia ją z wyrównanym do 4KB adresem fizycznym pamięci. Strukturą tą można manipulować poprzez instrukcje VMX takie jak VMPTLRD, VMCLEAR, VMREAD, VMWRITE. VMCS można podzielić na następujące części:

- Guest-state area - stan procesora zapisywany przy VM EXIT i ładowany spowrotem przy VM ENTRY (przejście w stan VMX non-root)
- Host-state area - stan procesora który jest ładowany przy VM EXIT
- VM-execution control fields - opisuje działanie procesora w trybie mniej uprzywilejowanym (non root), ma częściowy wpływ na sytuacje które powoduja VM EXIT
- VM-exit control fields - zawiera pola kontrolujace VM EXIT
- VM-entry control fields - odpowiednio kontroluje VM ENTRY
- VM-exit information fields - opisuje jakie zdarzenie spowodowało VM EXIT

2.4.2 Instrukcje VMX

VT-x wprowadza 10 instrukcji do kontroli maszyn wirtualnych:

- VMPTRLD - ładuje VMCS z pamięci
- VMPTRST - zapisuje VMCS w pamięci
- VMCLEAR - inicjalizuje region VMCS
- VMREAD - odczytuje z VMCS pole wyspecyfikowane przez rejestr źródłowy i kopiuje je do rejestru docelowego lub pamieci
- VMWRITE - zapisuje pole do VMCS z pierszego źródła (pamięć lub rejestr) wyspecyfikowane przez drugie źródło (rejestr)

- VMCALL - instrukcja używana przez “wirtualizowane” oprogramowanie do odwoływania sie do VMM (czyli spowodowanie VM EXIT)
- VMLAUNCH - ustawia stan VMCS na “launched” (o ile wcześniej stan był “clear”) i wykonuje VM ENTRY
- VMRESUME - o ile stan VMCS jest równy “launched” to wykonuje VM ENTRY
- VMXOFF - wyłącza instrukce VMX
- VMXON - włącza instrukcje VMX

2.5 AMD-V

Technologia AMD-V jest również znana jako AMD-SVM (Secure Virtual Machine) oraz pod nazwą kodową “Pacifica”. Jest to technologia wirtualizacyjna przeznaczona dla architektury AMD64 (x86_64). Pierwszymi procesorami ją obsługujacymi były Athlon 64 ("Orleans"), the Athlon 64 X2 ("Windsor") and the Athlon 64 FX ("Windsor").

Aby móc korzystać z technologii SVM, należy najpierw ustawić odpowiedni bit w EFER (Extended Features Register), inaczej wykonywanie instrukcji SVM będzie kończyć się niepowodzeniem.

2.5.1 VMCB

VMCB (Virtual Machine Control Block) to odpowiednik VMCS z technologii Intel VT-x. Podobnie jak w przypadku Intela VMCB może być utożsamiona tylko z wyrównanymi do 4K fragmentami fizycznej pamięci. Zasadniczo można ją podzielić na dwie części:

- Control Area - opisuje m.in które instrukcje mają być przechwytywane przez VMM, w jaki sposób ma się zachowywać procesor w przypadku VM EXIT i VM ENTRY
- State Save Area - przechowuje stan procesora w trybie mniej uprzywilejowanym (“maszyny wirtualnej”)

2.5.2 Instrukcje

- VMRUN - jest to centralna funkcja SVM. Uruchamia ona maszynę wirtualna na podstawie podanego w rejestrze rAX adresu w pamięci zawierającego zdefiniowaną strukturę VMCB. Stan procesora (tryb root) jest zapisywany w pamięci pod adresem zdefiniowanym w VM_HSAVE_PA MSR (Model-Specific Register)
- VMCALL - Podobnie jak w Intel VT-x, służy do odwoływania się z poziomu wirtualnej maszyny do VMM
- VMSAVE - zapisuje VMCB do pamięci
- VMLOAD- wczytuje VMCB z pamięci

2.6 Hardware versus Software

Mimo tego, że przejścia między "maszyną wirtualną" a maszyną natywną zostały zaimplementowane sprzętowo, VM EXIT i VM ENTRY potrzebują dużej ilości cykli, co przy częstych przechwytywaniach daje bardzo duży narzut czasu. Efekt jest taki, że na pierwszych procesorach obsługujących VT-x/SVM sprzętowa wirtualizacja sprawuje się znacznie gorzej niż metody softwarowe takie jak parawirtualizacja czy binarna translacja! Producenci procesorów próbuja więc zmniejszyć ilość cykli potrzebną na przełaczanie na VMM i z powrotem, oraz próbują zmniejszyć potrzebę interwencji ze strony monitora maszyn wirtualnych poprzez wprowadzanie "wirtualizacji" pamięci oraz operacji I/O.

3 Różne rodzaje obsługi tablic stron

Różne rodzaje wirtualizacji korzystają z różnych rodzajów stronicowania. Przykładowo - wirtualizacja software'owa, w tym translacja binarna, korzystają głównie z Shadow Page Tables, wirtualizacja wspomagana sprzętowo, korzystająca wydatnie z rozszerzeń procesora używa Nested Page Tables (przy procesorach AMD) czy też Extended Page Tables (ta sama zasada działania co Nested Page Tables, tylko używane na procesorach Intela). Parawirtualizacja, z racji tego, że modyfikuje system operacyjny gościa, może korzystać zarówno z Shadow Page Tables jak i Extended Page Tables. Czym więc się charakteryzują poszczególne podejścia?

3.1 Shadow Page Tables

System działający w wirtualnej maszynie posiada własne tablice stron (Guest Page Tables, w skrócie gPT). Oczywiście, nie może sięgnąć do pamięci bezpośrednio pod adres "fizyczny " na który wskazuje jego tablica stron, ponieważ nie jest to systemowy adres fizyczny. Dlatego, gdy gość chce sięgnąć do jakiejś strony w pamięci, przechwytuje to działanie hypervisor, który przy pomocy shadow page tables tłumaczy z adresów fizycznych gościa na adresy fizyczne systemu. Gość nie wie o tym, że hypervisor tłumaczy jego adresy, a hypervisor nie ma dostępu do adresów gościa. Żeby całe to tłumaczenie miało sens, musimy zapewnić zgodność sPT i gPT. Problemem są:

- modyfikacje w gPT (dodawanie, usuwnie translacji)
- błędy braku strony gość - host
- bity odwołania (accessed) w sPT
- w przypadku gości SMP - zgodność translacji adresów na wszystkich procesorach

3.1.1 Techniki utrzymywania zgodności sPT i gPT

1. Z wykorzystaniem błędów braku strony
    - Gość zmienia coś w gPT,
    - Następnie odwołuje się do właśnie zmienionego adresu w tablicy stron,
    - Procesor jest zmuszany do korzystania z sPT – nie ma tam jednak zmienionej translacji – generuje błąd braku strony,
    - Hypervisor przechwytuje ten błąd braku strony, dodaje do sPT brakującą translację i wykonuje instrukcję.
2. Ochrona gPT przed zapisem
    - Każde dodanie translacji kończy się jako wyjątek braku strony
    - Ten wyjątek jest przechwytywany przez hosta, który emuluje całą operację

- Analogicznie: usuwanie z gPT hypervisor usuwa/dodaje do gPT a potem update'uje sPT

Obie techniki powodują dużo błędów braku strony. Są dwa rodzaje takich błędów:

- wywołane przez gościa – guest induced – spowodowane normalnym działaniem gościa -muszą zostać obsłużone przez hypervisora
- wywołane przez shadow paging – hypervisor induced

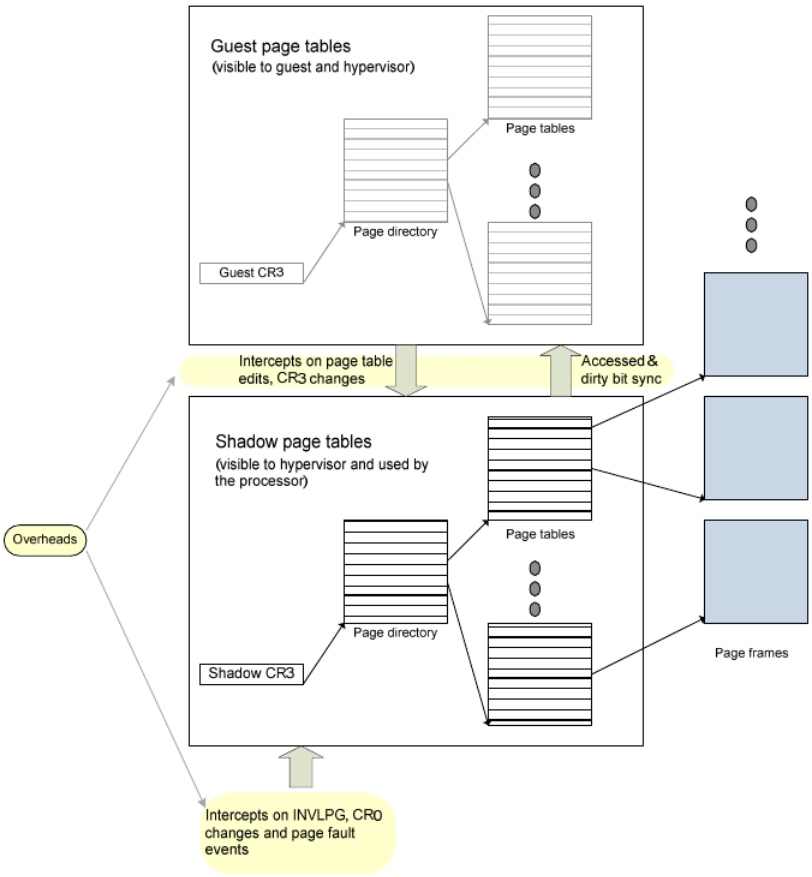


Figure 3: Guest and shadow page tables (showing two-level paging)

źródło: AMD-V Nested Paging White Paper [13]

3.1.2 Jak to wygląda w praktyce - XI Shadow Mechanism

Gość może zwiększyć dostęp do swoich stron zaznaczając strony, które w rzeczywistości są nieobecne jako obecne, a strony "read only" jako "read write". Hypervisor może odkryć, że jego sPT jest przestarzałe tylko wtedy, gdy gość ma błąd przy próbie dostępu lub zapisu na taką stronę. Analogicznie, by zmniejszyć dostęp do swoich stron, gość może zaznaczyć je jako nieobecne czy też "read only". Zanim jednak takie zmiany zostaną wprowadzone, gość musi zrobić TLB flush. To powoduje VMEXIT i hypervisor może znowu zsynchronizować swoje strony. W jaki sposób jednak hypervisor orientuje się, które tablice stron zostały zmienione? Przechodzenie przez wszystkie tablice stron gościa jest niepraktyczne, dlatego zazwyczaj trzyma się wszystkie tablice stron jako tylko do odczytu. Gdy gość chce coś zapisać, powoduje to błąd, a wtedy hypervisor dodaje taką stronę do listy niezsynchronizowanych i nadaje jej status do zapisu. Gdy gość robi TLB flush, hypervisor musi tylko zmienić status stron z listy niezsynchronizowanych. Jak już wszystkie zmiany zostaną wprowadzone, lista niezsynchronizowanych jest pusta, a wszystkie wpisy w gPT są znów tylko do odczytu.

3.1.3 Szeregowanie procesów w przypadku shadow paging

Gdy gość zmienia zawartość rejestru przechowującego adresy (CR3) hypervisor musi przechwycić tę operację, zmienić zawartość TLB i ustawić prawdziwą wartość CR3 (bazowaną na sPT a nie na gPT). Jest to bardzo kosztowne.

3.1.4 Shadow Paging w SMP

W gościu korzystającym z SMP ta sama instancja gPT może zostać użyta do translacji adresów na więcej niż 1 procesorze, ale hypervisor musi utrzymywać instancję sPT dla każdego z wirtualnych procesorów, lub współdzielić sPT pomiędzy wiele wirtualnych procesorów. Gdy dla każdego z wirtualnych procesorów utrzymywane jest sPT generuje to spore koszty pamięciowe, a gdy jest jedna sPT - mamy koszty synchronizacji. Całość stanowi ok. 75% kosztów generowanych przez hypervisora.

3.2 Nested Page Tables

Nested paging – zagnieżdżone stronicowanie – używa dodatkowej, albo zagnieżdżonej (nested) tablicy stron (nPT), przy pomocy której tłumaczy ad-

resy fizyczne gościa na systemowe adresy fizyczne. Zarówno gość jak i zarządca mają swoją własną kopię stanu procesora (części związanej ze stronicowaniem), a gość ma pełną kontrolę nad swoimi tablicami stron (gPT). Zarządca nie musi przechwytywać i emulować modyfikacji gPT. gPT tłumaczy adresy liniowe gościa na adresy fizyczne gościa. nPT tłumaczą adresy fizyczne gościa na adresy fizyczne widziane przez system. Gdy gość chce mieć dostęp do pamięci fizycznej, mechanizm przechodzi przez 2 tablice jednocześnie (gPT i nPT). Po takim przejściu TLB zawiera translację z adresu liniowego gościa na systemowy adres fizyczny i może zostać potem ponownie użyty. Nested paging pozostaje niezauważone przez gościa. Procesory AMD obsługujące NP np. używają takiej samej translacji niezależnie od tego, czy tłumaczą adresy gościa czy macierzystego systemu. Dodatkowo, procesory AMD wspierające zagnieżdżone stronicowanie utrzymują nested TLB, który zawiera translacje z adresów fizycznych gościa na adresy fizyczne systemu – przyspiesza to szukanie stron.

3.2.1 Koszty

Nietrafienie adresu w TLB ma większy koszt niż nietrafienie bez nested paging. Przykładowo, przy 4 poziomach stronicowania gości być może trzeba będzie przejść przez zagnieżdżoną tablicę stron 5 razy – po razie dla każdego gościa i raz dla ostatniej translacji stron danych gościa. Każde przejście może wymagać nawet do 4 dostępów do cache'a (tłumaczenie z adresu fizycznego gościa na adres fizyczny systemowy) i jeszcze jeden dostęp do pamieci by przeczytać wpis.

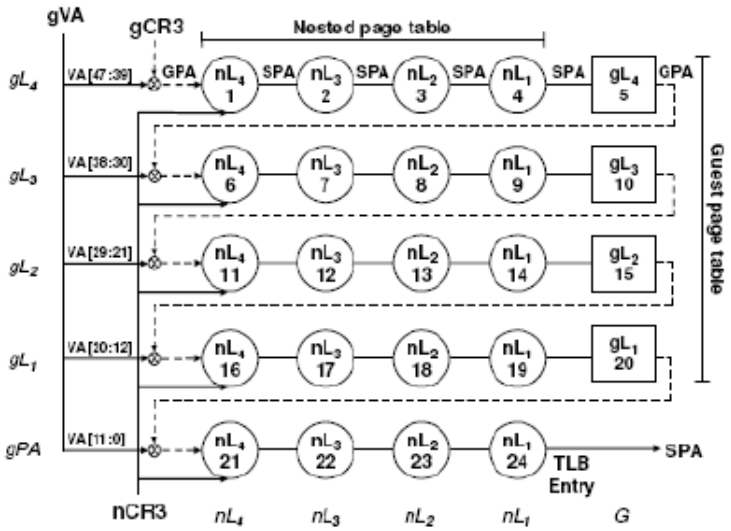


Figure 5: Address translation with nested paging. GPA is guest physical address; SPA is system physical address; nL is nested level; gL is guest level

źródło: AMD-V Nested Paging White Paper [13]

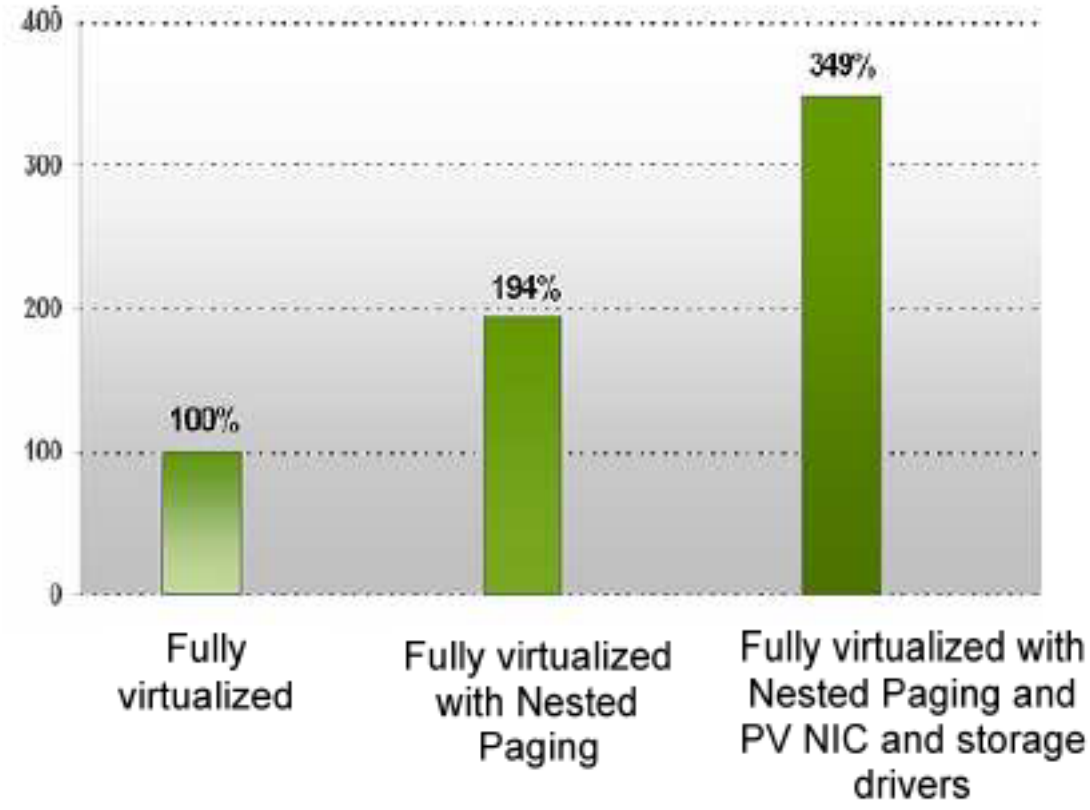
3.2.2 Oszczędności pamięci

- Wystarczy jedna instancja nPT by zmapować całą fizyczną przestrzeń adresową gościa. Przy shadow paging – dla każdej tablicy stron gościa (gPT) musiała istnieć tablica sPT (shadow page table)
- W przypadku SMP – wystarczy, że będzie jedna nPT - znacznie bardziej efektywne niż shadow paging – tam albo trzeba utrzymywać dla każdego procesora sPT (spory koszt pamięciowy) lub można utrzymać jedną, ale za to koszt synchronizacji jest duży
- Im większy rozmiar strony tym bardziej wydajność nested paging rośnie (bo większa strona oznacza, że mniej stron należy przejść by znaleźć wpis)
- Żeby poprawić wydajność nested paging, zarządca może wybrać jakie rozmiary stron ma mieć nPT. Dla AMD64 Quad-Core może być to 4KB, 2MB lub 1 GB

- tak samo, duży rozmiar strony gościa zmniejsza koszt nietrafienia w TLB

3.2.3 Inne cechy nested paging

Nested paging zostało wprowadzone w rodzinie procesorów AMD Barcelona. W Xen 3.0.5 – również możliwe, także możliwe jest tam shadow paging. Zagnieżdżone stronicowanie jest włączane przy uruchamianiu i wspiera wszystkie tryby stronicowania.



źródło: AMD-V Nested Paging White Paper [13]

Na obrazku przykład wydajności: Oracle 10G OLTP z oraz bez nested paging z RHEL 4.5 pod Xen na AMD64 Quad-Core Model 2350.

4 Wirtualizacja urządzeń wejścia wyjścia

4.1 Problemy związane z wirtualizacją urządzeń wejścia wyjścia

Nie można pozwolić na bezpośredni dostęp do urządzeń I/O, bo wtedy maszyny wirtualne mogą sobie wzajemnie „grzebać" w pamięci => bardzo wolne. Jednocześnie, chcemy by wszystkie maszyny wirtualne miały dostęp do urządzeń wejścia/wyjścia. Wirtualizacja urządzeń wejścia/wyjścia musi spełnić pewne wymagania, tzn. musi być w stanie wykryć urządzenie, kontrolować je, a także przesyłać dane i obsługiwać przerwania pochodzące od urządzeń.

4.2 Możliwe rozwiązania

4.2.1 Emulacja

Jest to obsługa sprzętu i wielu wbudowanych opcji z poziomu oprogramowania. Jej zaletą jest to, że nie wymaga zmian w systemie gościa, ale VMM musi udostępniać dokładnie takie opcje jak sprzęt – zawiera kompletną implementację urządzenia. Jest to przenośne.

4.2.2 Parawirtualizacja

VMM wystawia API dla sterowników urządzeń w systemie operacyjnymgościu, które są zmodyfikowane.Zaletą jest lepsza wydajność, a wadą – konieczność modyfikacji gościa.

4.2.3 Bezpośredni przydział (direct assignment)

Polega na tym, że maszyna komunikuje się bezpośrednio ze sprzętem. Jego wadą jest to, że można przydzielić tylko tyle urządzeń ile jest ich rzeczywiście, a gdy chcemy przenieść maszynę wirtualną na inny serwer musimy zapewnić dokładnie takie same urządzenia. Główną zaletą jest to, że jest to szybsze niż emulacja i jest mniej kodu obsługi urządzeń w VMM.

4.3 Input-Output Memory Management Unit

IOMMU Łączy szynę DMA z urządzeń wejścia wyjścia z pamięcią główną, zajmuje się tłumaczeniem widocznych dla urządzeń adresów wirtualnych

(zwanych także adresami urządzeń) na adresy fizyczne. Może też zajmować się ochroną pamięci.

4.3.1 Przykłady występowania IOMMU:

- Graphics Address Remapping Table używane przez karty AGP i PCI Express.
- W procesorach AMD = „Hyper Transport Architecture"
- W procesorach Intel – Virtualization Technology for Directed I/O, w skrócie VT-d
- IOMMU – firmy Sun
- IBM – Translation Control Entry

4.3.2 Różne implementacje IOMMU

1. $< arch/x86_64/kernel/pci - nommu.c >$ - nie używać IOMMU w ogóle (np. bo jest $<$ 3GB pamięci).
2. $< arch/x86_64/kernel/pci - gart.c >$: sprzętowe IOMMU bazujące na AMD GART
3. $< arch/x86_64/kernel/pci - swiotlb.c >$ - software'owa implementacja IOMMU. Używana gdy nie ma sprzętowego IOMMU w systemie a jest potrzebne, bo komputer posiada >3GB pamięci, albo kernel ma włączoną opcję iommu=soft.
4. $< arch/x86_64/pci - calgary.c >$ - sprzętowe IOMMU IBM Calgary. Używane w serwerach pSeries i xSeries. To IOMMu wspiera mapowanie adresów DMA z ochroną pamięci.

4.4 IOMMU na przykładzie AMD I/O virtualization technology

IOMMU opiera się na koncepcji „domen ochronnych" (protection domains). Każde urządzenie I/O zostaje przypisane do konkretnej domeny i wybranego zbioru tablic stron. Gdy urządzenie chce coś wczytać lub zapisać do pamięci

systemowej IOMMU sprawdza przy pomocy domeny i tablic stron, czy urządzenie ma do tego prawo oraz gdzie w pamięci znajdują się strony do których chce mieć dostęp. Dlatego IOMMU składa się z:

- Tablic stron urządzeń wejścia/wyjścia, które IOMMU używa do sprawdzania uprawnień i tłumaczenia adresów dla pamięci, do której urządzenia mają dostęp
- Tablicy urządzeń, w której jest przypisanie urządzenia do konkretnej domeny i która zawiera wskaźniki do tablicy stron I/O
- Interrupt remapping table (tablicy ponownego mapowania pamięci w wypadku przerwań), która jest używana do sprawdzania uprawnień i przemapowania pamięci w przypadku przerwań od urządzeń

4.4.1 Plusy IOMMU

- Duże obszary ciągłej pamięci mogą być zaalokowane bez konieczności zapewniania ich ciągłości w pamięci fizycznej
- Niektóre urządzenia nie są w stanie zaadresować całej przestrzeni adresowej – IOMMU robi to za nie
- Ochrona pamięci przed źle działającymi urządzeniami – takie urządzenie nie może mazać po pamięci, która nie została mu przydzielona

4.4.2 Minusy IOMMU

- większy koszt przechodzenia po tablicy stron
- większe zużycie pamięci – IOMMU dodaje tablicę stron dla urządzeń wejścia/wyjścia

4.4.3 IOMMU i wirtualizacja

Oprócz wymienionych powyżej cech IOMMU, które również dotyczą wirtualizacji, ułatwia ona DMA. DMA w maszynach wirtualnych jest utrudnione, bo gość zapewnia fizyczne adresy wirtualne, a urządzenie, które rozpoczyna DMA nie wie o tym, że te adresy są jeszcze tłumaczone przez macierzysty system operacyjny, który przechwytuje operację wejścia wyjścia, co spowalnia tę

operację. W przypadku, gdy używamy IOMMU – można przemapować pamięć tak, by sterowniki do urządzenia z systemu operacyjnego zarządcy były użyte przez gościa. U Intela – Virtualization Technology for Directed I/O (Intel VT-d) w ramach IOMMU jest wsparcie dla przemianowania transferów DMA (np. można przypisać urządzenie bezpośrednio do konkretnej maszyny wirtualnej i użyć sterowników z macierzystego systemu operacyjnego) oraz obsługa przerwań sprzętowych – lepsza izolacja urządzeń wejścia/wyjścia.

5 Uproszczenie wirtualizacji dzięki wsparciu sprzętowemu

5.1 Xen i KVM

Xen i KVM potrafią na procesorach z wsparciem dla wirtualizacji emulować sprzęt bez uciekania się do parawirtualizacji oraz tłumaczenia binarnego. Tak robi to KVM:

- Kod systemu gościa wykonuje się w ring 0 , programy gościa w ring 3, programy gościa działają zupełnie normalnie.
- Linuxa (z modułem KVM, będący nadzorcą) pracuje w ring -1, gdy następuje odwołanie do systemu gościa zostaje ono przekazane przez niego do ring 0.
- Kod systemu gościa wykonuje się, aż do momentu, gdy wyda rozkaz uprzywilejowany. Zostaje on przechwycony przez procesor i przekazany do KVM. On z kolei wie, co się stało i robi pracę za system gościa, być może musi przekazać prace do ring 3 do qemu które symuluje urządzenie wejścia/wyjścia i zwrócić to do gościa.
- KVM jest mały (co do ilości kodu) i łatwy do zarządzania (nie trzeb dużo zmieniać w Linuxie bo dużo rzeczy dzieje się w ring 3, a mało w ring -1). Niemniej nadal potrzeba shadow page tables oraz nadal zmiany w nich są drogie.
- Maszyna wirtualna to w gruncie rzeczy zwykły proces, którym można normalnie zarządzać (z tym, że z mojego doświadczenia <kvm-75 na amd64> wynika, iż proces ten może łatwo spowodować krytyczny błąd jądra w ring -1!!).

- Shadow page tables to najbardziej skomplikowana część całego KVM, resztą w zasadzie zajmują się wirtualne urządzenia qemu.

Xen:

- Oczywiście musi dodatkowo implementować shadow page tables.
- Do wirtualizacji I/O używa qemu, ale jest to proces bardziej skomplikowany niż w KVM:
  - Trzeba przeskoczyć do specjalnej domeny z qemu (domena to pewna maszyna wirtualna).
  - Przedtem jednak Xen spyta system zarządzania czasem, czy gościowi należy się czas procesora, potem dopiero przełączy się do qemu. Ono emuluje urządzenie, rozpocznie transfer.
  - Znowu nastąpi skok do Xen. Dopiero teraz skok z powrotem do gościa, który dostaje dane wygenerowane przez qemu.
  - Przewaga Xen – można na maszynie gościa zainstalować specjalne sterowniki do I/O – sam kod jądra nie jest modyfikowany, dodajemy tylko wydajne sterowniki do I/O, które są przystosowane do Xen i dlatego działają bardzo szybko.
  - Zarządzanie pamięcią działa podobnie jak w KVM i VmWare.

Należy pamiętać ze VMENTRY i VMEXIT są bardzo drogie i robią TLB FLUSH (no chyba, że mamy najnowsze procesory – AMD PHENOM albo CORE i7 – ja takich nie posiadam) - dlatego KVM i Xen nie zawsze wygrywają w testach szybkości, gdy używamy gościa o kodzie niezmodyfikowanym do pracy jako maszyna wirtualna.

5.2 Co dalej?

Przyszłość wirtualizacji należy prawdopodobnie do systemów łączących parawirtualizacje z pełną wirtualizacją, wykorzystując do tego wsparcie sprzętowe. Już teraz na Xen możne się wykonywać system nieświadomy działania w wirtualnej maszynie, można jednak do takiego systemu zainstalować sterowniki, które będą efektywnie współpracowały z uproszczonym interfejsem urządzeń wirtualnych – łączymy zalety obydwu podejść do wirtualizacji. Nad podobnym rozwiązaniem pracuje Red Hat w KVM.

6 Zagrożenia związane ze sprzętową wirtualizacją. Użycie sprzętowej wirtualizacji do tworzenia niewykrywalnego szkodliwego oprogramowania (BluePill, Virtriol i podobne).

Wirtualizacja sprzętowa z jednej strony zwalnia monitora maszyny wirtualnej (VMM) z robienia wielu nieistotnych z jego punktu widzenia rzeczy, takich jak obsługa syscalli z procesów gościnnego systemu operarcyjnego, czy "dopasowanie" fizycznych tablic stron do ich "wirtualnych" odpowiedników. Z drugiej strony stwarza zupełnie nowe zagrożenia. Jest to na swój sposób zaskakujące, gdyż wyrafinowane mechanizmy sprzętowe zwykle są wprowadzane po to, aby chronić przed szkodliwymi działaniami oprogramowania.

Przykładem "nadużycia" są tak zwane Hardware Virtual Machine Rootkits (albo ogólniej Virtual Machine Based Rootkits, w skrócie VMBR). Rootkit to w skrócie narzędzie pomocne we włamaniach do systemów informatycznych. Pozwala on na ukrycie plików i procesów, które umożliwiają utrzymanie jakiejś formy kontroli nad opanowanym systemem. Hardware VM rootkits wykorzystują nowe instrukcje wprowadzone na procesorach AMD i Intela, i ukrywając się przy pomocy "sprzętowych" metod, czynią się niezwykle trudnymi do wykrycia.

6.1 Przykłady VM Based Rootkits

Najbardziej znane VMBR to:

- Blue Pill - rootkit autorstwa Joanny Rutkowskiej z COSEINC (obecnie z Invisible Things). Oryginalnie został napisany pod Windows Vista x64 z użyciem technologii AMD SVM ("Pacifica"). Nowa wersja tego VMBR (New Blue Pill), udostępniona publicznie ma też wsparcie dla Intel VT-x.

- SubVirt - VMBR autorstwa naukowców z Microsoft Research oraz Uniwersytetu Michigan. Pozwala on na załadowanie oryginalnego systemu operacyjnego "do wewnątrz" Microsoft Virtual PC. Nie jest to de facto Hardware VM rootkit, ale warto o nim wspomnieć z tego względu, że najprawdopodobniej stanowił inspirację do powstania Blue Pill.

- Vitroil - autorstwa Dino Dai Zovi z Matasano. Rootkit pod MacOS X używający Intel-VT-x.

6.2 Jak działają VM Based Rootkits?

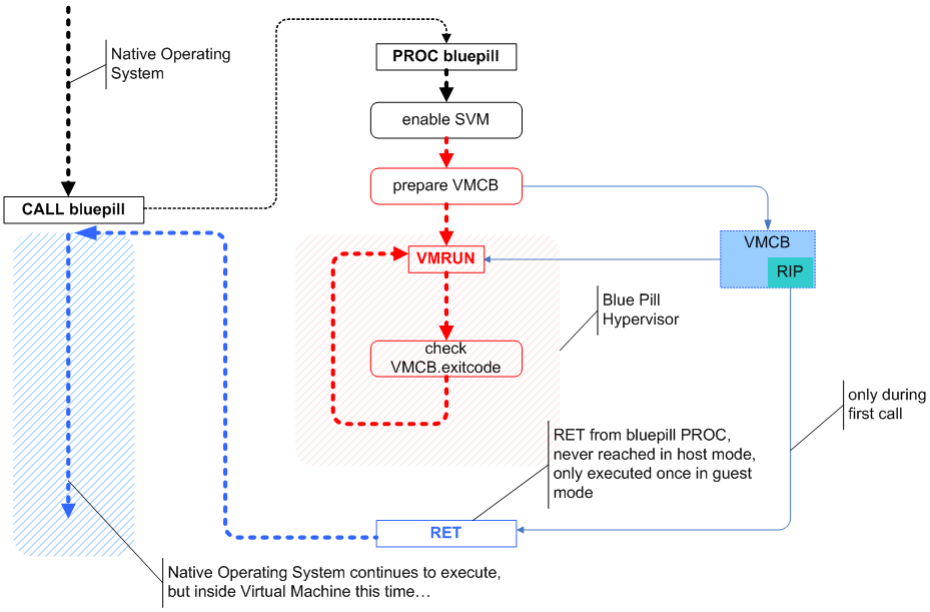
1. Startują w trybie jądra (ring 0)
2. Instalują swojego nadzorcę maszyny wirtualniej (Hypervisor). W przypadku HVM rootkits polega to na alokacji pamięci dla "lekkiego" nadzorcy (Thin Hypervisor) i załadowanie go.
3. Przenoszą obecnie uruchomiony system operacyjny na wirtualną maszynę ("OS połyka niebieską pigułkę")
4. Przechwytują "interesujące" zdarzenia (w przypadku "sprzętowych" rootkitów polega to zwykle na obsługiwaniu VM EXIT)

6.2.1 Różnice w działaniu: SubVirt i Blue Pill

SubVirt	Blue Pill
Jest zainstalowany "na stałe", musi bowiem dokonać "przenosin" podczas bootowania sie oryginalnego systemu. Można go więc łatwo wykryć "Offline" (kiedy komputer jest wyłączony).	Instaluje się "w locie", bez reboota i zmianach w BIOSie i na dysku. Nie można go wykryć "Offline". Oznacza to również, że przestaje on działać po reboocie systemu, ale to nie jest problem z jego punktu widzenia: serwery mają długi uptime, a użytkownicy laptopów cześciej używają hibernacji niż zamykania systemu
Działa na x86, gdzie nie ma pełnej wirtualizacji	Działa korzystając z technologii AMD SVM która, jak zapewniają jej twórcy, zapewnia pełną wirtualizację
Bazuje na komercyjnym VMM, który sam emuluje sprzęt	Dzięki sprzętowej wirtualizacji nadzorca Blue Pill jest niewielki, a korzystanie ze sprzętu nie powoduje zmniejszenia wydajności

Zastosowanie sprzętowych metod powoduje, że Hardware VM rootkits takie jak np Blue Pill są praktycznie niemożliwe do wykrycia bezpośrednimi metodami, i trzeba się uciekać do metod pośrednich.

6.2.2 Przyklad "przenoszenia" OS-a na wirtualną maszynę (Tutaj Blue Pill i AMD SVM)



źródło: Subverting Vista (TM) Kernel For Fun And Profit [9]

6.3 Jak można wykrywać rootkity tego rodzaju?

6.3.1 Rożne zachowanie się instrukcji wewnątrz i na zewnątrz VM

Niestety znalezienie takiej instrukcji oznaczałoby że wirtualizacja nie jest pełna - sprzeczność z założeniem twórców takich technologii jak AMD SVM.

6.3.2 Próba utworzenia maszyny wirtualnej wewnątrz maszyny wirtualniej

Taka próba powinna wywołać VM EXIT i przy złej obsłudze zakończyć sie niepowodzeniem

- Niestety, można to łatwo obejść, wystarczy że rootkit jest w stanie obsłużyć "zagnieżdzone" maszyny wirtualne (coś takiego potrafi New Blue Pill).

6.3.3 Testy czasowe

Wykorzystywany jest fakt, że niektóre instrukcje muszą wywołać VM EXIT. Na przykład dla Blue Pill na procesorach AMD taką instrukcją musi być RDMSR EFER (Read from Model Specific Register) - pozwalająca na sprawdzenie czy bit SVM w rejestrze EFER (Extended Features Register) jest włączony. "nadzorca" musi bowiem "oszukiwać" system operacyjny na wirtualnej maszynie, że ten bit jest wyłączony. normalne wykonanie tej instrukcji trwa około 90 cykli procesora, w przypadku przechwytywania przez VMM czas wydłuża się do 2000 cykli

- W przypadku używania do pomiarów czasu licznika czasu procesora (np. używając instrukcji RDTSC), HPET (High Precision Event Timer) lub RTC (Real Time Counter) VMM może oszukać detektor maszyny wirtualnej poprzez technikę "śledzenia RDTSC", przechwytywanie przerwań lub przechwytywanie I/O)

6.3.4 Testy czasowe z zewnętrznym zegarem

Jest to ten sam sposób, jak testy czasowe, ale wykorzystujemy zewnętrzne źródło czasu ("zegarek w ręku", szyfrowane połączenie z serwerem czasowym NTP)

- Nie ma ogólnej metody na oszukanie takiego źródła czasu. Ale można zastosować "Strategię kurczaka", która polega na wykryciu częstych wywołań instrukcji wywołujących VM EXIT (czyli wykrycie potencjalnego detektora) i chwilowe odinstalowanie się "w locie" z możliwością ponownego zainstalowania się za jakiś czas.

6.3.5 Testy licznikowe

Podobne do testów czasowych, lecz do mierzenia "czasu" służy wątek wykonywany na innym procesorze (zwykła pętla z licznikiem)

- Wymaga wielordzeniowego procesora. Nie można zastosować "strategii kurczaka". Nie można też oszukać licznika, chyba że przez coś podobnego do binarnej translacji.

6.3.6 "Zapychanie" TLB

TLB (Translation Lookaside Buffer) jest zapełniane znanymi detektorowi adresami w pamieci. poprzez wymuszenie VM EXIT (np. wspomniana wyżej instrukcja RDMSR EFER) niektóre wpisy w TLB zostaną zamazane (zniszczone) więc wykrycie zniszczonych asocjacji (poprzez mierzenie czasu dostępu) pozwoli na stwierdzenie, że HVM rootkit jest uruchomiony.

- Bardziej zagmatwana wersja rzeczywiście jest w stanie wykrywać np. Blue Pilla.

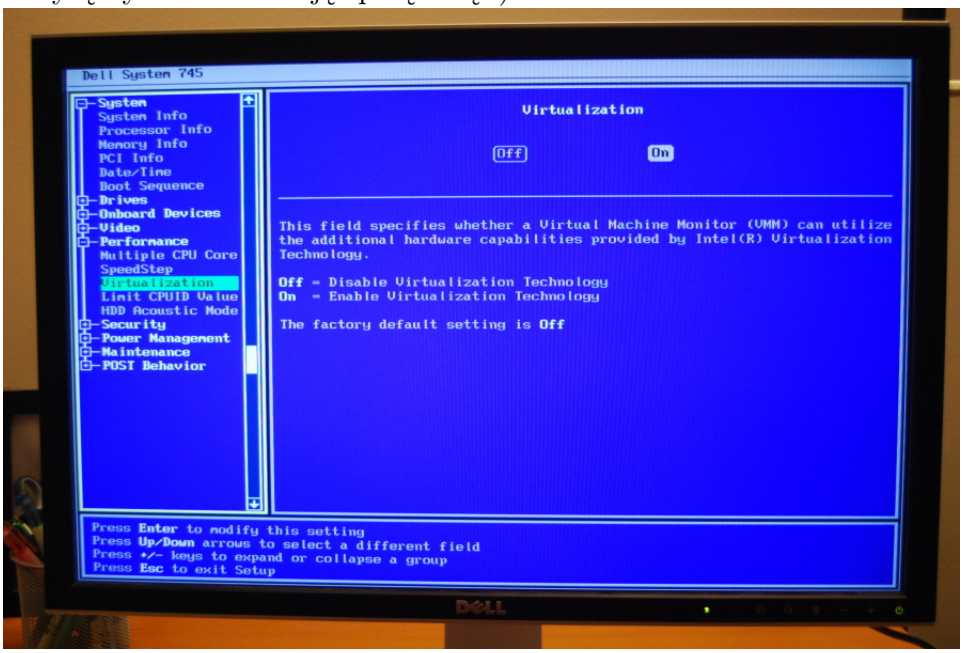
6.3.7 "Adams' pill"

Jak wyżej opiera się również na zapychaniu TLB, ale wykorzystuje inne sposoby na to żeby wykryć wpisy TLB po VM EXIT,VM ENTRY

- Z powodu specyficznego działania cache L2, który też trzeba zapełnić, trzeba wygenerować wirtualne adresy o okreslonym wzorze. VMM może wykryć takie działanie, i zastosować "Strategię kurczaka"

6.4 A może po prostu...

- ...wyłączyć wirtualizację sprzętową :)



- ...wymyśleć dodatkowe instrukcje, np instrukcję sprawdzającą: SVMCHECK - otrzymywałaby ona hasło, które byłoby inne dla każdego procesora - to zapobiegałoby pisaniu programów które inaczej działają na wirtualnej maszynie i na fizycznej maszynie. Sprawdzałaby ona oczywiście czy jesteśmy wewnątrz VM czy nie.

7 Bibliografia

1. http://cache-www.intel.com/cd/00/00/19/76/197666_197666.pdf
2. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pd
3. http://en.wikipedia.org/wiki/Hardware-assisted_virtualization
4. http://en.wikipedia.org/wiki/Timeline_of_virtualization_development
5. http://en.wikipedia.org/wiki/X86_virtualization
6. http://pl.wikipedia.org/wiki/Intel_Virtualization_Technology
7. http://pl.wikipedia.org/wiki/Tryb_wirtualny http://en.wikipedia.org/wiki/Virtual_8086_n
8. http://en.wikipedia.org/wiki/Blue_Pill_(malware)
9. http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf
10. http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zovi.pdf
11. http://bluepillproject.org/stuff/IsGameOver.ppt
12. http://www.amd64.org/fileadmin/user_upload/pub/2007XenSummit-AMD-Barcelona_Nested_Paging_WahligHuang.pdf
13. http://developer.amd.com/assets/NPT-WP-1\%201-final-TM.pdf
14. http://software.intel.com/en-us/articles/best-practices-for-paravirtualization-enhancements-from-intel-virtualization-technology-ept-and-vt-d
15. http://en.wikipedia.org/wiki/IOMMU
16. http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art02.pdf
17. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/344
18. http://blog.egenera.com/2008/02/io-virtualization-defined/
19. http://lxr.linux.no/linux+v2.6.27/Documentation/x86/x86_64/boot-options.txt\#L219
20. http://en.wikipedia.org/wiki/Ring_(computer_security)

21. http://it.anandtech.com/IT/showdoc.aspx?i=3263&p=1

22. http://www.virtualization.info/2008/03/intel-to-introduce-new-virtualization.html