Tablice stron, bezpośredni dostęp do urządzeń z maszyny wirtualnej i IOMMU.

Filip Grotkowski

21 listopada 2008

Omówienie problemów i ich rozwiązań.

Spis treści

1 Tablice stron

1.1 Shadow Page Table

Techniki softwarowe wymagają utrzymywania kopii (shadow version), tablicy stron gościa (gPT). Kiedy gość jest aktywny, podczas wykonywania translacji adresów, hypervisor nakazuje procesorowi działanie na tych właśnie kopiach. Głównym problemem związanym z utrzymywaniem Shadow Page Table (sPT) jest oczywiście spojność tej struktury z tablicą stron gościa. Istnieje wiele rozwiązań tego problemu. Jednym z nich jest ochrona przed zapisem fizycznych stron, które stanowią gPT. Każda operacja dodania translacji skutkuje wyjątkiem page fault. Następnie procesor oddaje sterowanie do hypervizera, a ten emuluje odpowiednie operacje. Podobnie hypervizor przejmuje kontrolę kiedy gość edytuje gPT w celu usunięcia translacji. W tym przypadku translacja jest usuwana z gPT, a następnie są przeprowadzane aktualizacje w sPT.
Inna metoda stronicowania w tle nie zabezpiecza przed nadpisaniem stron stojących za gPT, ale za to polega na sposobie w jaki procesor zachowuje się przy obsłudze błędów odwołań do tablicy stronicowania oraz stosowaniu reguł spójności TLB (w kontekście danego gościa). W tej technice zwanej czasem Virtual TLB, hypervizor pozwala gościowi dodawać nowe translacje do gPT bez wnikania w te operacje. Dopiero jeśli gość, wykonując jakieś operacje, będzie chciał odwołać się do pamięci poprzez dodaną translację, procesor spowoduje page fault (bo translacja nie jest wpisane do sPT). W konsekwencji page fault pozwoli hypervizorowi zainterweniować, tzn. przejżeć gPT, dodać do sPT brakującą translację i wykonać instrukcje, która spowodowała błąd. Podobnie jeśli gość będzie chciał usunąć translację. Wtedy wykona on INVLPG (usunięcie wpisu w TLB), by unieważnić translację w TLB, a hypervizor usunie odpowiednią translację z sPT i wykona INVLPG (Invalidate Translation Look-Aside Buffer Entry - usunięcie wpisu w TLB) dla usuniętej translacji.
Obie techniki owocują w wiele page faultów. Już zwykłe zachowanie gościa przynosi dużo błędów - np. sięganie po strony, które zostały już odesłane do pamięci przez SO gościa i trzeba je ściągać od nowa. Takie błędy nazywamy indukowanymi przez gościa page faultami, muszą być one analizowane przez hypervizor i odsyłane z powrotem do gościa, co powoduje zwiększenie kosztów takiej operacji w stosunku do natywnego stronicowania. Page faulty spowodowane przez sPT nazywamy indukowanymi przez hypervizor. Dla odróżnienia dwóch powyższych page faultów, hypervizor musi przejść przez sPT i gPT, co znacznie zwiększa koszty.
Kiedy gość jest aktywny, page walker ustawia bity accessed i dirty w sPT. Ale skoro gość polega na dobrych ustawieniach tych bitów w gPT, hypervizor musi odzwierciedlać je w gPT. Dla przykładu, gość może użyć tych bitów do stwierdzenia, które strony mogą być przeniesione na dysk, by zwolnić miejsce dla nowych stron.
W sytuacji kiedy gość reshedule'uje nowy proces, odświeża rejestr CR3, by gPT odnosiło się do nowego procesu. Hypervizor także bierze udział w tej operacji unieważniając wpisy w TLB związane z poprzednią zawartością CR3 i ustawia nową wartość CR3 bazującą na sPT dla nowego procesu. Znów widać, że contex switch'e u gościa podnoszą koszty.
Utrzymywanie sPT poważnie spowalnia czas reakcji maszyny wirtualnej i zwiększa zużycie pamięci dla gości SMP. W przypadku takich gości, to samo gPT może być użyte do translacji adresów dla więcej niż jednego procesora. Od hypervizora wymagamy wtedy by utrzymywał

odpowiednią liczbę sPT (dla każdego procesora) lub by dzielił sPT pomiędzy wiele wirtualnych procesorów. Dodatkowo wiemy, że większe zużycie pamięci w przyszłości pociąga za sobą zwiększenie kosztów synchronizacji.
Szacuje się, że aż do 75% normalnego działania hypervizora może pochłaniać stronicowanie w tle.

1.2 AMD Nested Page Table

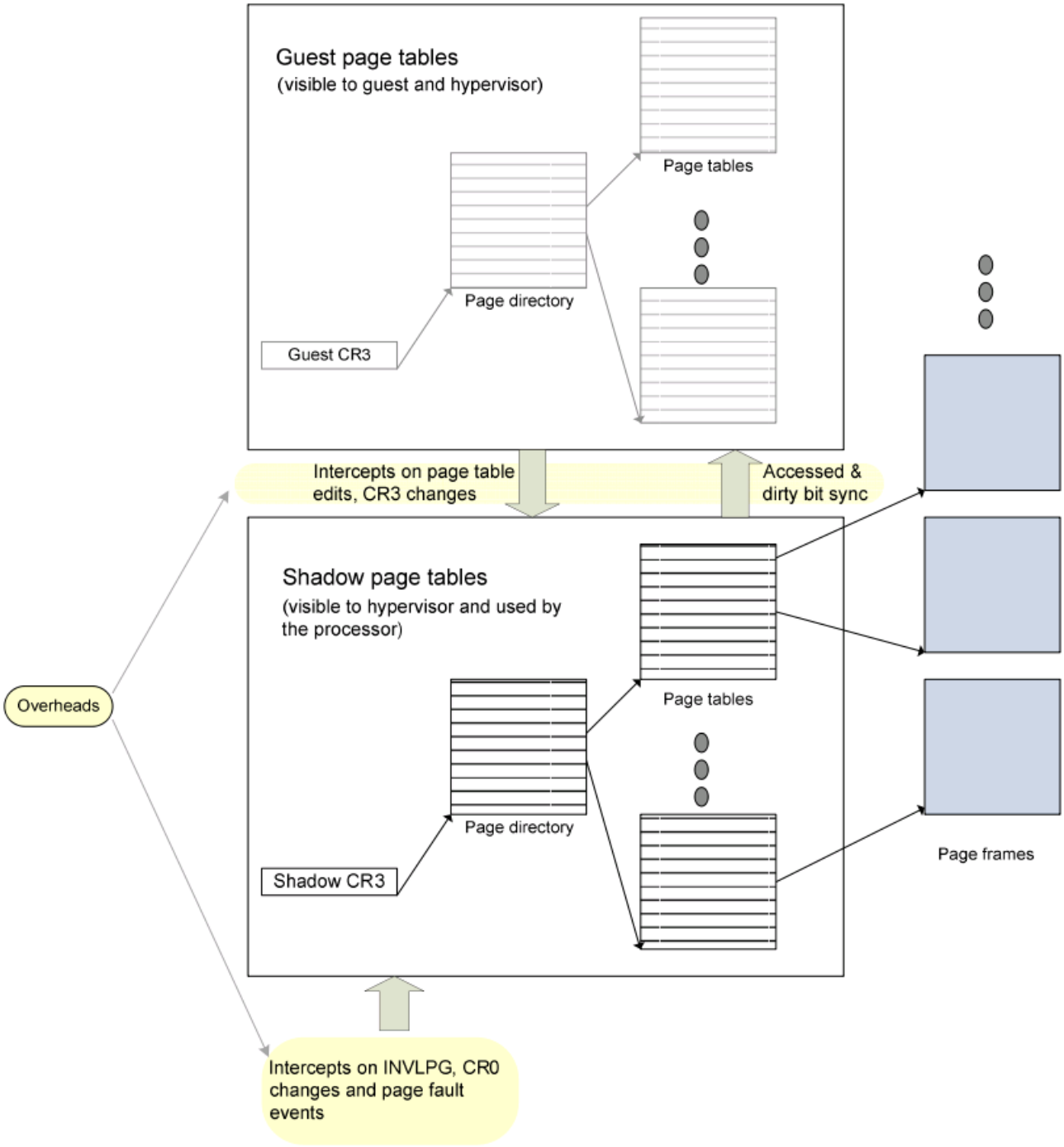
1.2.1 Wprowadzenie

Aby uniknąć kosztów narzucanych przez oprogramowanie stronicowania w tle, procesory AMD64 QuadCore mają dodany Nested Paging do sprzętowego translatora adresów. Nested Paging używa dodatkowych tablic stron (NPT), by tłumaczyć fizyczne adresy gościa na fizyczne adresy hosta. Zostawia przy tym gościowi całkowitą kontrolę nad jego tablicami stron. Inaczej niż jak przy shadow page'ingu, raz stworzone dodatkowe tablice stron, nie wymagają później od hypervizora żadnych interwencji, czy też emulacji zmian wprowadzanych przez gościa w gPT. Zatem Nested Paging niweluje koszty związane ze stronicowaniem w tle, ale ponieważ wprowadza dodatkowy poziom translacji, szansa nie trafienia adresu w TLB się zwiększa.

1.2.2 Szczegóły

Oboje gość i hypervizor mają swoje własne kopie danych stanu procesora dotyczących stronicowanie, np. CR0, CR3, CR4 (ang. Control Register), EFER i PAT (Page-Attribute Table). gPT tłumaczy liniowe adresy gościa na jego fizyczne adresy. Nested Page Table (nPT) tłumaczy adresy fizyczne gościa na adresy fizyczne hosta.

Dodatkowe tablice stron (nested) i tablice stron gościa są utrzymywane przez odpowiednio hypervizor i gościa. Kiedy gość spróbuje odwołać się do pamięci posługując się liniowym adresem i nested paging jest włączony, page walker wykonuje 2 wymiarowy/stopniowy algorytm używający gPT i nPT do tłumaczenia adresu liniowego gościa do adresu fizycznego systemu (Rysunek 2).



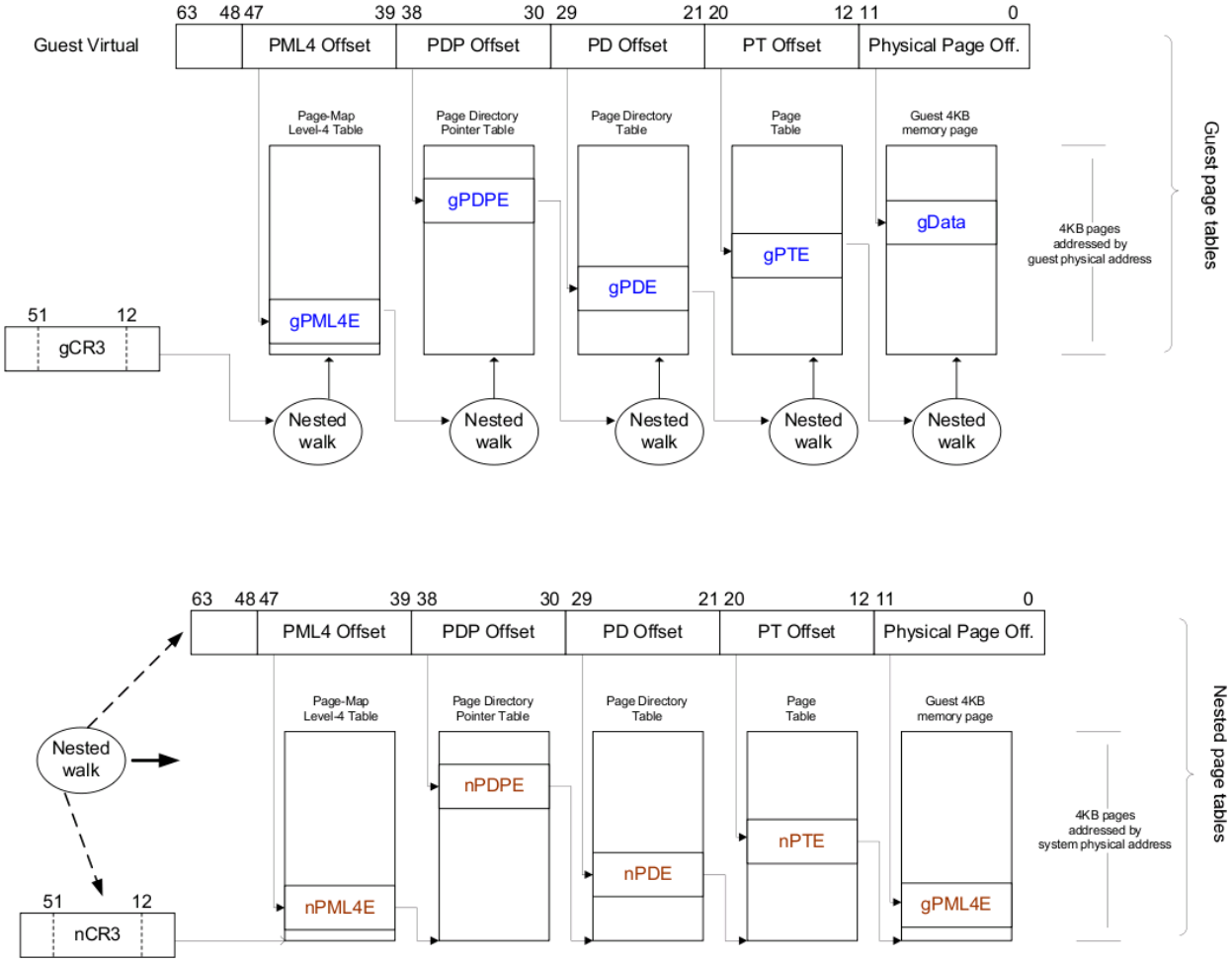
Rysunek 1: stronicowanie gościa i w tle (dwa poziomy stronicowania).

Kiedy powyższa operacja się zakończy, TLB entry zawierające translacje z liniowego adresu gościa do fizycznego adresu hosta jest cache'owany w TLB i może być używany do dalszych odwołań do wspomnianego adresu liniowego. Procesory AMD wspierające nested paging używają tego samego TLB niezależnie od tego, czy procesor jest w trybie gościa czy hosta (hypervizora). Zatem gdy procesor jest w trybie gościa, TLB ma wpisy typu adres liniowy gościa i odpowiadający mu adresu fizyczny systemu. A gdy procesor jest w trybie hosta, TLB ma wpisy typu adres liniowy hosta i odpowiadający mu adres fizyczny systemu.

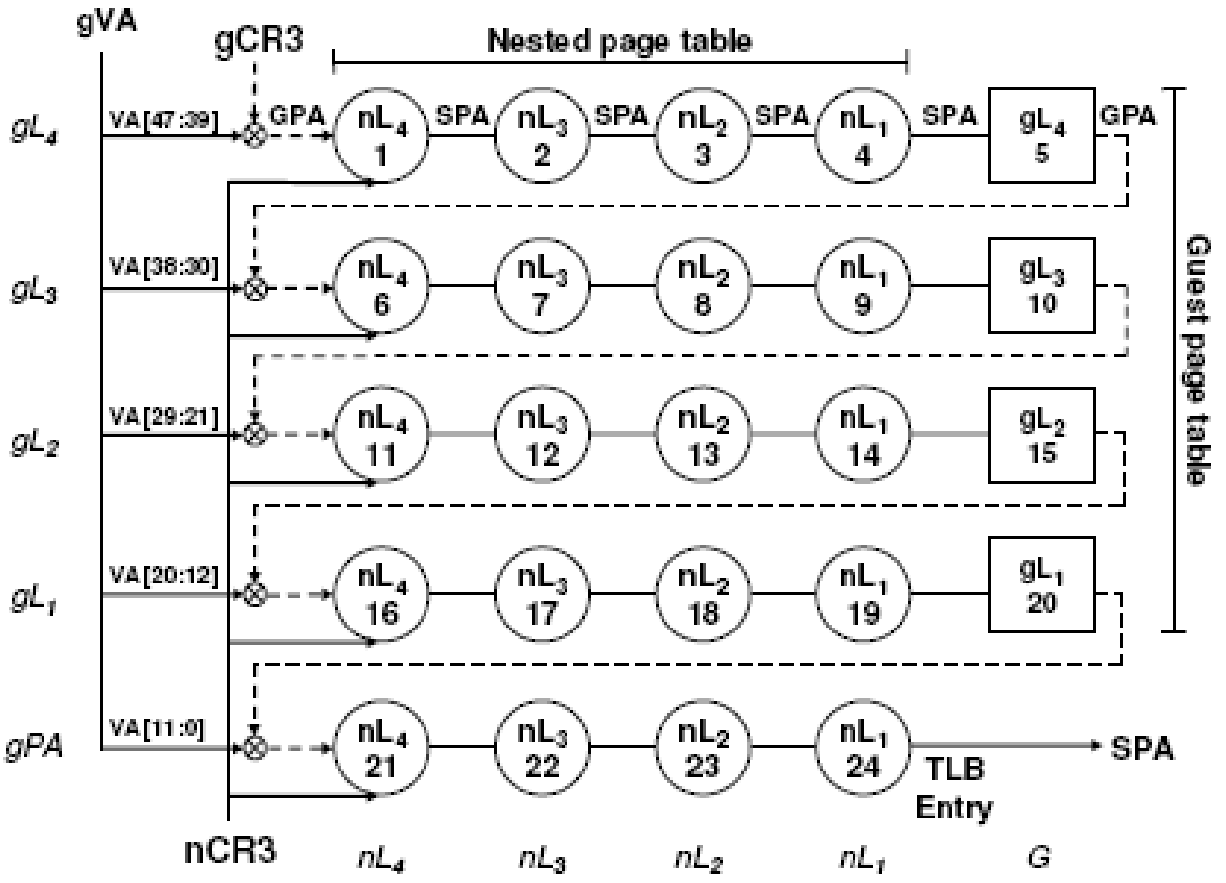
Dodatkowo procesory AMD, które wspierają nested paging, by przyspieszyć przejście po tablicy stron (page walk), posiadają Nested TLB, który cache'uje translacje typu: adres fizyczny gości i odpowiadający mu adres fizyczny systemu.

1.2.3 Koszt translacji

Nie trafienie szukanego wpisu w TLB jest przy nested page'ingu bardziej prawdopodobne niż bez niego. A to dlatego, że przy nested page'ingu tłumaczenie adresów wymaga nie tylko chodzenie po gPT, ale również równolegle chodzenie po nPT w celu tłumaczenia adresów fizycznych gościa (pojawiających się podczas chodzenia po gPT, np. gCR3 lub adresów stojących za wpisami w gPT) na adresy fizyczne systemu. Przechodząc do konkretów, czteropoziomowe wyszukiwanie powoduje pięciopoziomowe wyszukiwanie w nPT (mianowicie czterokrotne tłumaczenie każdego fizycznego adresu gościa i jedno końcowe tłumaczenie szukanego adresu).



Rysunek 2: Tłumaczenie adresu liniowego gościa na adres fizyczny systemu z wykorzystaniem nested page tables.



Rysunek 3: Translacja adresów z nested page'ingiem. GPA to adres fizyczny gościa, SPA, to adres fizyczny systemu, nL(G) to poziom nested (gościa).

Zatem każde tłumaczenie wykonywane na nPT wymaga do czterech sięgnięć do pamięci, by przetłumaczyć adres fizyczny gościa na systemowy adres fizyczny, oraz jednego by dostać się do szukanego miejsca. Jak widać, gdy TLB nie jest skuteczne koszt nam wzrasta z 4 sięgnięć do pamięci, do 24 takich sięgnięć, przy odpowiednio wyłączonym i włączonym nested page'ingu (Rysunek 3).

Nested paging jest funkcjonalnością przeznaczoną do użytku dla hypervizora. Gość nie widzi przy tym żadnej różnicy działając pod hypervizorem korzystającym z nested page'ingu. W szczególności nie trzeba wprowadzać żadnych modyfikacji u gościa (tak, jak to ma miejsce w przypadku np. para-wirtualizacji). NPT jest opcjonalną cechą, nie obejmującą wszystkich procesorów wspierających AMD-V. Oprogramowanie może skorzystać z instrukcji CPUID (stwierdzającej jakie funkcje są wspierane przez procesor) by rozpoznać czy nested page'ing jest wspierany przy konkretnym procesorze.

1.2.4 Oszczędność pamięci

W odróżnieniu od shadow-page'ingu, który wymaga od hypervizora utrzymywania sPT dla każdej instancji gPT, hypervizor korzystający z nested page'ingu zadowala się tylko jedną instancją nPT, która pokrywa całą przestrzeń adresową gościa (fizyczną). Z uwagi na zwartość pamięci gościa nPT powinna z reguły zajmować mniej pamięci niż odpowiednia implementacja shadow-page'ingu.

Przy nested page'ingu hypervizor może używać jednej instancji nPT jednocześnie, na więcej niż jednym procesorze przy SMP. Takie podejście jest dużo bardziej efektywne w porównaniu z implementacją opartą na shadow page'ingu, gdzie hypervizor powoduje większe zużycie pamięci (utrzymując sPT dla każdego procesora) lub wydłuża synchronizację (sPT trzeba modyfikować, a nPT nie).

1.2.5 Wpływ rozmiaru strony na szybkość nested page'ingu

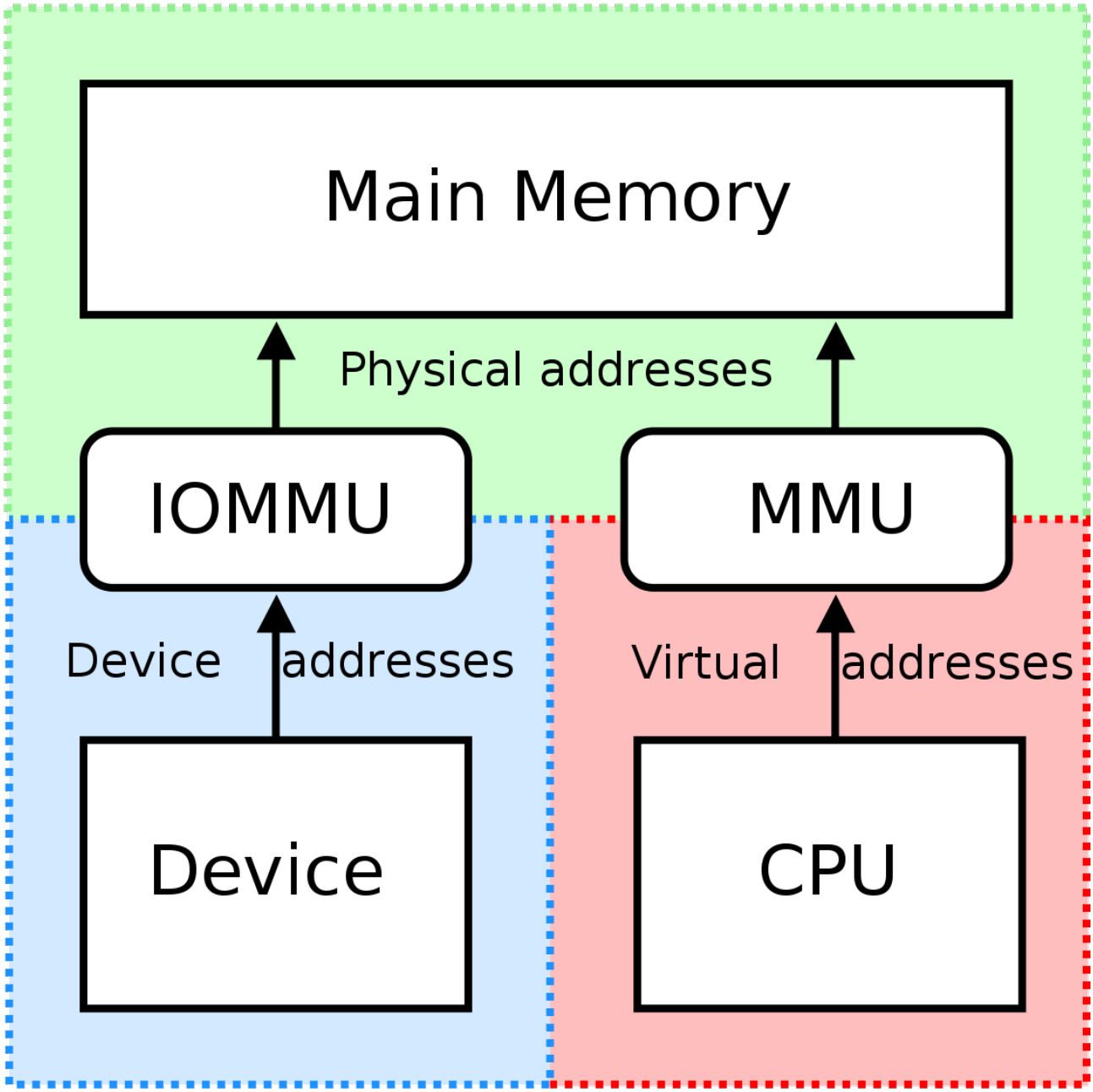
Oczywistym jest, że tłumaczenie adresów jest szybsze jeśli TLB myli się stosunkowo rzadko. Czynnikiem wpływającym na prawdopodobieństwo trafienia szukanego adresu w TLB rośnie jeśli mamy do przejścia mniej stron (mniej poziomów). Zatem duże strony zmniejszają liczbę poziomów, które musimy przejrzeć podczas tłumaczenia adresów. Aby przyspieszyć nested page'ing, hypervizor może wybrać nPT oparte na różnych rozmiarach stron. Ponadto rozmiary nPT są różne dla różnych gości i mogą się zmieniać podczas działania gościa. I tak procesor AMD64 Quad-Core daje do dyspozycji trzy rozmiary jednej strony: 4KB, 2MB i 1GB. Efektem ubocznym dużego rozmiaru strony jest skuteczność TLB. Przy większych stronach, każdy wpis w TLB pokrywa większą liczbę tłumaczeń adresów, tym samym zwiększając pojemność TLB i redukując czas translacji.

1.3 Intel Extended Page Table

2 Bezpośredni dostęp do urządzeń z maszyny wirtualnej

3 IOMMU

Input/output memory management unit (IOMMU) to MMU, które łączy szynę IO, korzystającą z DMA z pamięcią główną. Podobnie jak MMU, które tłumaczy adresy widziane przez procesor na adresy fizyczne, IOMMU tłumaczy adresy wirtualne widziane przez urządzenia (w tym kontekście urządzenia we/wy) na adresy fizyczne. IOMMU daje także pewne możliwości ochrony przed niebezpiecznymi zachowaniami urządzeń. Przykładem IOMMU jest Graphics Address Remapping Table (GART), używany przez AGP i PCI Express, które to ładują do pamięci kształty, siatki wielościanów, tekstury i inne dane za pomocą DMA.



Rysunek 4: Porównanie MMU i IOMMU.

3.1 IOMMU a wirtualizacja

Podczas działania SO wewnątrz maszyny wirtualnej, włączając systemy korzystające z parawirtualizacji (typu Xen), nie wie on jakiego właściwie adresu fizycznego pamięci używa. To czyni bezpośredni dostęp do sprzętu trudnym. Jest tak, ponieważ kiedy SO, próbuje zlecić sprzętowi bezpośrednie sięgnięcie do pamięci (DMA), może to zniszczyć pamięć, bo sprzęt nie wie o dodatkowym tłumaczeniu między liniowym, a fizycznym adresem gościa. Ten problem jest rozwiązywany przez hypervizor lub SO hosta, który interweniuje podczas operacji we/wy, tłumacząc adresy. Niestety taka operacja jest kosztowna i spowalnia we/wy. Z pomocą przychodzi jednak IOMMU, które może dokonać dodatkowego tłumaczenia adresów, do których chce się odwołać sprzęt. Dzieje się to z wykorzystaniem, tej samej lub podobnej tablicy translacji, której używa gość.

3.2 AMD IOMMU

3.3 Intel VT-d

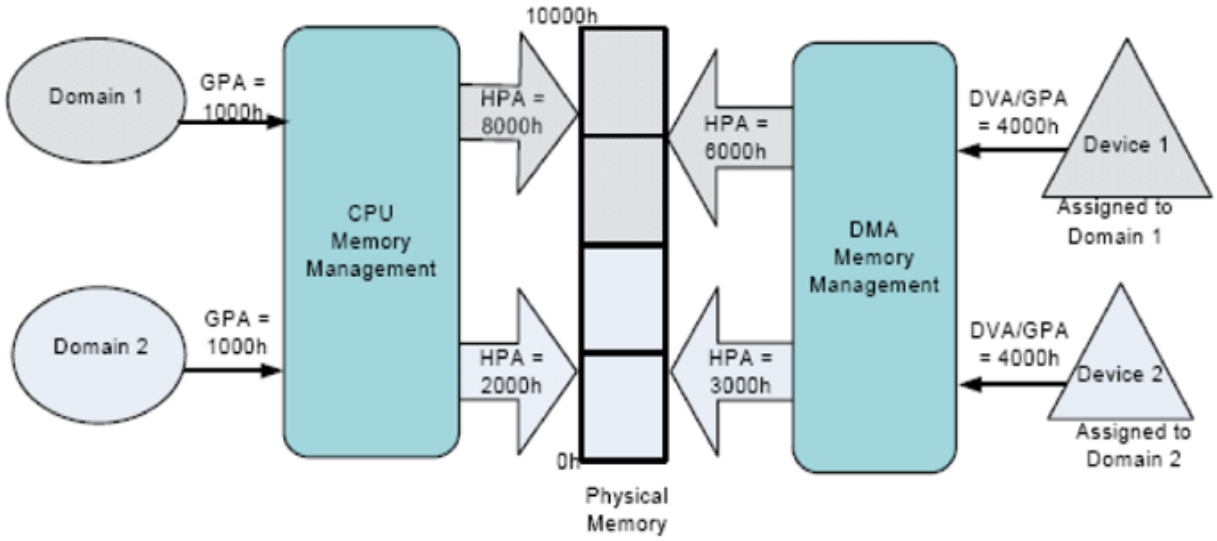
Jak można wyczytać na stronie producenta: Intel VT-d umożliwia zwiększenie stabilności, elastyczności oraz wydajności I/O w środowisku zwirtualizowanym, poprzez umożliwienie bezpośredniego podłączania urządzeń I/O do maszyn wirtualnych.

Przechodząc do szczegółów, architektura Intel VT-d jest uogólnieniem architektury IOMMU, która pozwala oprogramowaniu SO, na tworzenie wielu chronionych dziedzin DMA (DMA protection domain). Chroniona dziedzina DMA, jest abstrakcyjnie zdefiniowanym środowiskiem, w którym jest zaalokowana część pamięci fizycznej hosta. W zależności od modelu oprogramowania, chroniona dziedzina DMA może reprezentować pamięć zaalokowaną dla wirtualnej maszyny, pamięć DMA zaalokowaną przez sterownik SO gościa, działającego na wirtualnej maszynie, lub część samego hypervizora. Architektura Intel VT-d pozwala oprogramowaniu SO przypisać jedno lub więcej urządzeń we/wy do chronionej dziedziny. Izolacja DMA jest osiągnięta za pomocą restrykcyjnych sposobów dostępu do pamięci fizycznej chronionej dziedziny, przez urządzenia we/wy nie przypisane do niej (dba o to tablica translacji adresów).

Urządzenie we/wy przypisane do chronionej dziedziny może widzieć adresy innymi, niż widzi jest host. VT-d traktuje adresy wyspecyfikowane w żądaniu DMA, jako wirtualne adresy DMA (DVA). Zależnie od modelu oprogramowania, DVA może być GPA maszyny wirtualnej, do której przypisane jest urządzenie, lub jakimś abstrakcyjnym adresem wirtualnym we/wy (podobnym do liniowego adresu procesora). VT-d przekształca adres żądania DMA wydanego przez urządzenie we/wy, w odpowiedni adres fizyczny hosta.

3.3.1 Rzutowanie adresów na chronione dziedziny

Aby wspierać wiele chronionych dziedzin, sprzęt tłumaczący żądania DMA musi identyfikować urządzenie je zlecające. Identyfikator wysyłanego zlecenia jest złożony z szyny PCI / urządzenia / numeru funkcji przypisanego przez oprogramowanie konfiguracyjne PCI, i jednoznacznie identyfikuje funkcję sprzętu, która nadesłała zgłoszenie.



Rysunek 5: Ilustracja translacji DMA przy współistnieniu wielu dziedzin. Urządzenia 1 i 2 są przypisane do chronionych dziedzin 1 i 2, odpowiednio, każde ze swym własnym obrazem przestrzeni adresowej DMA.

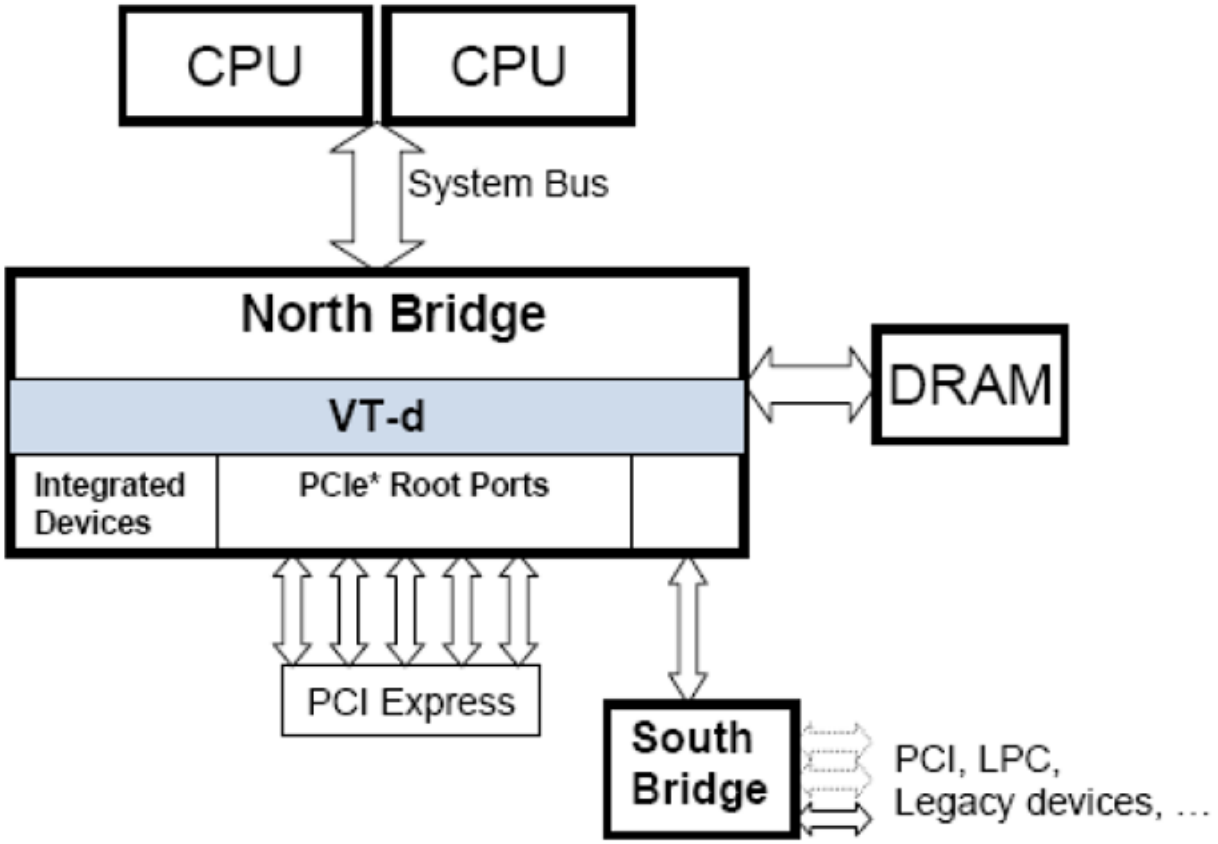
Architektura VT-d definiuje struktury danych potrzebne do mapowania urządzeń we/wy do dziedzin ochrony (Rysunek 8):

1. Root-Entry Table: Funkcjonuje jako struktura najwyższego poziomu, do tłumaczenia urządzeń na szyny PCI. Część z numerem szyny z id żądania jest używana do indeksowania root-entry table. Każde root entry zawiera wskaźnik do context-entry table.

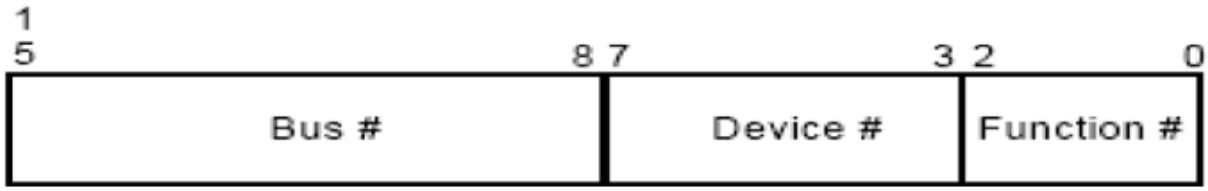
2. Context-Entry Table: Każdy wpis w tej tablicy rzutuje urządzenie we/wy na danej szynie, na chronioną dziedzinę, do której jest ono przypisane. Część id z urządzeniem i numerem funkcji jest używana do indeksowania tej tablicy. Każdy wpis context entry zawiera wskaźnik do struktur tłumaczenia adresów, używanych do tłumaczenia adresów w żądaniach DMA.

3.3.2 Tłumaczenie adresów

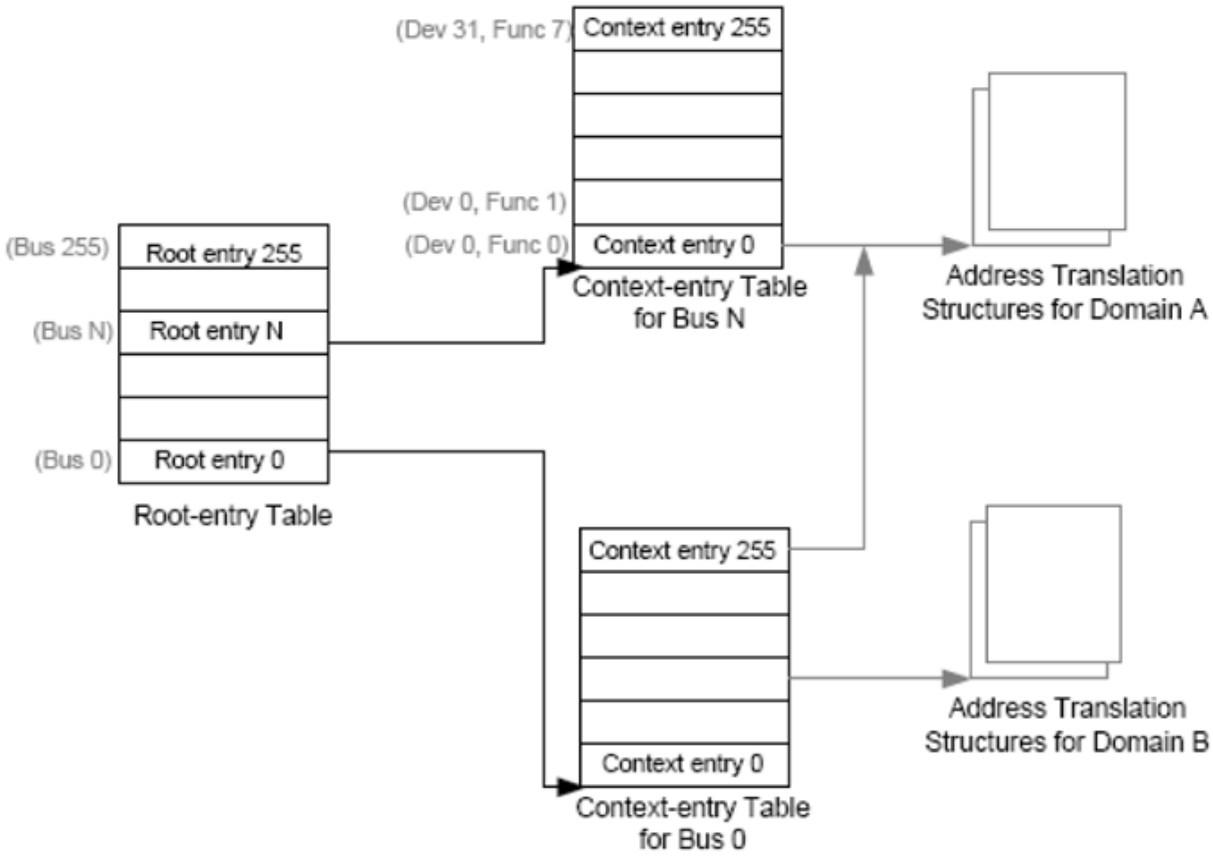
Architektura VT-d definiuje wielopoziomowe struktury tablic stron do tłumaczenia adresów DMA (Rysunek 9). Tablice stron, o których mowa są bardzo podobne do tych z procesora IA-32, pozwalając oprogramowaniu na używanie 4KB lub większych rozmiarów stron. Sprzęt implementuje przeszukiwanie tablic stron na podstawie adresu rządania DMA. Liczba poziomów, które trzeba pokonać, przechodząc przez tablice stron jest określona poprzez context-entry odpowiadające korzeniowi tablicy stron. Katalog stron i wpis w tablicy stron specyfikują niezależnie prawa do czytania i pisania, a sprzęt oblicza narastające prawa pisania i czytania napotkane podczas



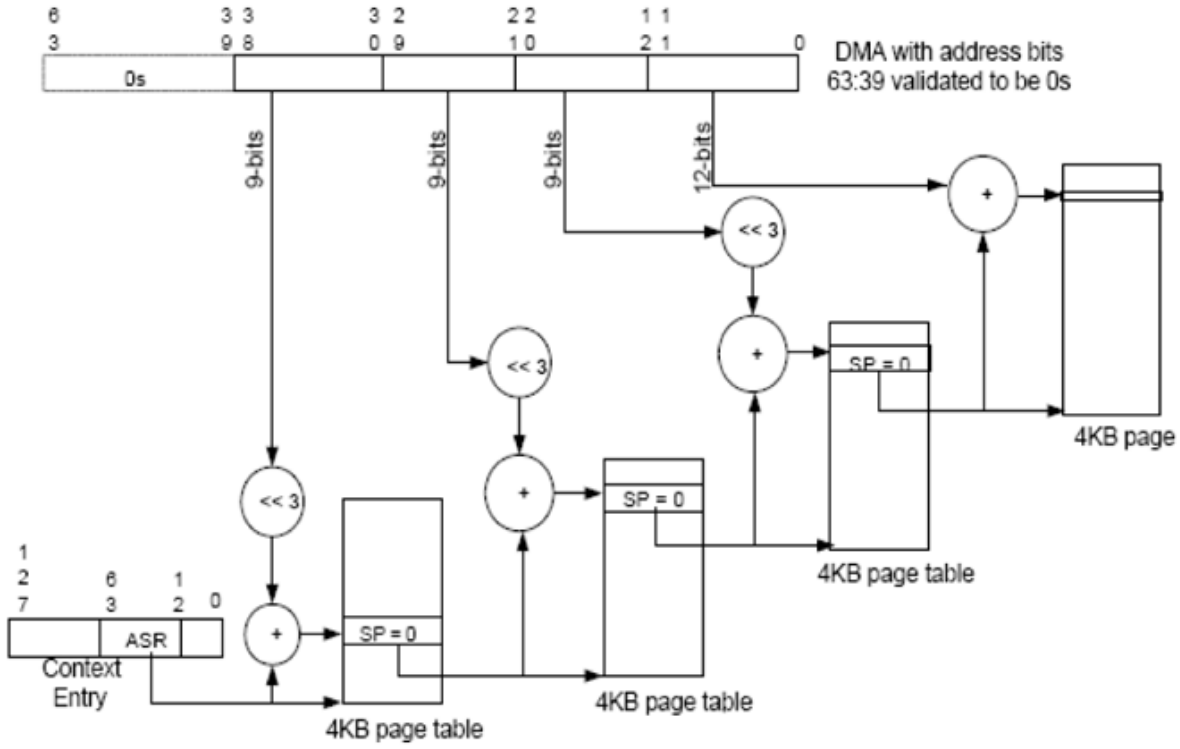
Rysunek 6: Ilustruje konfiguracje PC w VT-d wkomponowanym w mostek północny.



Rysunek 7: Format żądania PCI.



Rysunek 8: Struktury danych tłumaczenia urządzeń na dziedziny ochrony.



Rysunek 9: 3 poziomowa tablica stron.

przeszukiwania jako efektywne prawa dla żądań DMA. Struktury katalogów stron i tablic stron są stałego rozmiaru 4KB, ale większe rozmiary możemy także uzyskać (poprzez tzw. super-page support).

3.3.3 Rzutowanie VT-d DMA

Rzutowanie (remaping) Intel VT-d DMA pozwala zredukować wystąpienia VM exit dla urządzeń, które są przypisane do jakiejś dziedziny. żądanie DMA specyfikuje id zleceniodawcy i adres, zaś sprzęt tłumaczy te informacje na fizyczny dostęp do pamięci, używając do tego odpowiednich struktur danych.

Rzutowanie DMA wprowadza izolację, sprawdzając, czy ID żądającego ma dostęp do adresów i tłumaczy dostarczony przez urządzenie adres na pamięć fizyczną. Rzutowanie DMA jest stosowalne do wszystkich zasobów DMA, i wspiera także istniejący sprzęt.

3.3.4 Rzutowanie przerwań VT-d

Rzutowanie Intel VT-d pozwala zredukować koszty wirtualizacji przerwań dla urządzeń, które są przypisane do jakiejś dziedziny. Przerwanie specyfikuje id zleceniodawcy i id przerwania, zaś

sprzęt tłumaczy te informacje na przerwanie fizyczne, używając do tego odpowiednich struktur danych (Interrupt Remap Table) mieszczących się w pamięci.

Rzutowanie przerwań wprowadza izolację, sprawdzając, czy ID żądającego jest do tego uprawnione i generuje przerwanie z atrybutami z odpowiedniej struktury. Sprzet cache'uje najczęściej używane struktury, a oprogramowanie może dynamicznie odświeżać wpisy, dla efektywniejszego obsługiwania przerwań. Rzutowanie przerwań jest stosowalne do wszystkich źródeł przerwań, włączając przerwania otrzymywane przez starsze urządzenia.

3.3.5 Zastosowanie

Mając do dyspozycji technologie Intel VT-d, hypervizor może przypisać kartę graficzną bezpośrednio jednej maszynie wirtualnej. W takiej sytuacji system operacyjny gościa ma pełną kontrolę nad urządzeniem. Co więcej, może dodatkowo używać własnych akceleratorów (czyli funkcji API wspomagających generowanie grafiki, np. OpenGL i DirectX). To wszystko, w ramach maszyny wirtualnej, daje użytkownikowi pełne możliwości, porównywalne z natywnymi (włączając grafikę 3D). Z kolei nie mając wsparcia Intel VT-d, karta graficzna jest emulowana w hypervizorze poprzez oprogramowanie i żadne akceleracje (typu OpenGL lub DirectX) nie są możliwe.

W dzisiejszych czasach ten przykład może przemawiać do wyobraźni. Użytkownik używając wielu aplikacji na różnych systemach operacyjnych jednocześnie, nie chce poświęcać graficznych możliwości swojego komputera na wirtualizacje. Mając np. dwa systemy operacyjne działające na jednym sprzęcie równocześnie, dodatkowo każdy SO ma swój procesor, można korzystać z obu równocześnie, nie martwiąc się o spadek jakości grafiki.

3.3.6 Cache'owanie wpisów struktur

Aby przyspieszyć operacje rzutowania żądań DMA i przerwań, architektura VT-d jest wyposażona w pamięć podręczną, zapamiętującą najczęściej odwiedzane wpisy używanych struktur danych.

1. Context Cache: cache'uje najczęściej używane wpisy tłumaczące urządzenia na chronione dziedziny.

2. PDE (Page Directory Entry) Cache: cache'uje najczęściej używane wpisy w katalogu stron napotkane przy translacji.

3. IOTLB (I/O Translation Look-aside Buffer): cache'uje używane efektywne adresy (wyniki translacji).

4. Interrupt Entry Cache: cache'uje wpisy w strukturze interrupt-remapping table.

3.4 Zastosowania IOMMU poza wirtualizacją

IOMMU oprócz kluczowej w tym referacie kwestii wirtualizacji, ma także inne zastosowania. Należą do nich:

1. Duże obszary pamięci mogą zostać zaalokowane, bez potrzeby ciągłości w pamięci fizycznej. IOMMU zajmuje się mapowaniem ciągłych obszarów pamięci wirtualnej do pofragmentowanych obszarów pamięci fizycznej. W konsekwencji, czasami nie jest potrzebne użycie techniki wektorowego we/wy (ang. Vectored I/O lub scatter/gather I/O, czytanie lub pisanie z/do wektora buforów - odbywa się sekwencyjnie).

2. Dla urządzeń, które nie wspierają odpowiednio długiego adresowania, by zaadresować całą pamięć fizyczną, IOMMU pozwala ten problem rozwiązać. Dodatkowo unikamy przy tym kosztów związanych z kopiowaniem z i do buforów przestrzeni adresowej, którą urządzenie potrafi zaadresować.

   Przykład:
   Obecne maszyny x86 mogą używać więcej niz 4 GiB pamięci (opcja PAE w procesorach x86). Jednak, w dalszym ciągu zwykłe 32-bitowe urządzenie PCI nie potrafi zaadresować pamięci powyżej granicy 4 GiB, a więc nie może skorzystać z DMA. Bez IOMMU SO jest zmuszony implementować pochłaniające czas tzw. double buffers (Windows) lub bounce buffers (Linux) (chodzi o sytuację, gdy wykonujemy DMA I/O z lub do high memory, wtedy musimy zaalokować pamięć w low memory zwaną właśnie bounce buffers, z lub do której najpierw dane są kopiowane).

3. W pewnych architekturach IOMMU zajmuje się także w tłumaczeniu przerwań sprzętowych. Jest to robione analogicznie do standardowego tłumaczenia adresów pamięci.

4. IOMMU może również wspierać stronicowanie pamięci peryferyjnej. Urządzenie peryferyjne korzystając z PCI-SIG PCI Address Translation Services (ATS) Page Request Interface (PRI) może wykryć i zasygnalizować potrzebę zarządzania pamięcią menadżerowi usług (manager services).

5. Z myślą o niepożądanych zachowaniach urządzeń, nie mogą one pisać/czytać do/z pamięci, która nie została im do tego celu przypisana. Ochrona pamięci bazuje na fakcie, że SO działający na procesorze ma na wyłączność obie jednostki: MMU i IOMMU. W rezultacie urządzenia są fizycznie powstrzymywane przed zmianami w tablicach organizujących pamięć (właśnie za pośrednictwem tych jednostek).

   Z wirtualizacją, SO gościa może użyć sprzętu, który nie jest bezpośrednio dla niego przeznaczony. Wysoko wydajny sprzęt, jak np. karta graficzna, używa DMA, by sięgać do pamięci. W środowisku wirtualnym wszystkie adresy pamięci są tłumaczone przez oprogramowanie maszyny wirtualnej, co powoduje, że DMA nie działa. IOMMU obsługuje tę translację, pozwalając na używanie przez SO gościa natywnych sterowników urządzeń.

   W systemach z architekturą, w której port we/wy jest w innej przestrzeni adresowej niż pamięć, IOMMU nie jest używane kiedy procesor komunikuje się z urządzeniami poprzez port we/wy. W systemach z architekturą, w której port we/wy i pamięć są w jednej przestrzeni adresowej, IOMMU może tłumaczyć operacje dostępu do portów we/wy.

Literatura

[1] I/O Virtualization and AMD's IOMMU

[2] AMD-VTM Nested Paging

[3] Best Practices for Paravirtualization Enhancements from Intel Virtualization Technology: EPT and VT-d

[4] AMD I/O Virtualization Technology (IOMMU) Specification License Agreement

[5] http://en.wikipedia.org/wiki/IOMMU

[6] Intel Virtualization Technology for Directed I/O