Sprzętowe wsparcie dla wirtualizacji

Paweł Dziepak

Instytut Informatyki Uniwersytetu Wrocławskiego

10 grudnia 2014

Maszyny wirtualne

Co chcemy osiągnąć?

- izolacja – maszynie wirtualnej nie ufamy tak samo jak zwykłym procesom
- wirtualizacja zamiast emulacji – czym wiecej da się zrobić sprzętowo tym lepiej
- kompatybilność – parawirtualizacja to jednak trochę za mało

Wirtualizacja – co jest potrzebne?

...a raczej co chcielibyśmy mieć:

- zarządzanie stanem vCPU
- wirtualizacja uprzywilejowanych instrukcji (niektórych)
- zarządzanie pamięcą należącą do VM (np. zagnieżdżone stronicowanie)
- wirtualizacja przerwań
- komunikacja ze sprzętem (IOMMU, emulacja)

Host i guest

x86

VMX root – host (*Virtual Machine Monitor* w języku Intela). **VMX non-root** – guest.

aarch64

EL2 – hypervisor. **EL1** – kernel. **EL0** – aplikacje użytkownika.

Przejścia między trybami: **VM entry** - nieciekawe, **VM exit** - ciekawe, wpływa na wydajność VM.

Stan powiązany z VM

- rejestry – ogólnego przeznaczenia, instruction pointer, coś od TLS - jak w wątkach
- rejestry – przestrzeń adresowa, konfiguracja przerwań - prawie jak w procesach
- rejestry – różne mniej lub bardziej specyficzne dla konkretnej architektury rejestry dostępne zwykle tylko w trybie uprzywilejowanym (np. MSR w x86)
- stan VM – VM aktywna/nieaktywna
- stan CPU – aktywny, już nieaktywny, jeszcze nieaktywny, śpi

Stan rejestrów $\times$ 2 - host i guest.

x86 - wszystko w VMCS (*Virtual Machine Control Structure*). aarch64 - porozrzucane po wielu rejestrach.

Konfiguracja VM

- które instrukcje powodują VM exit (*trapping* w języku ARM)
- które przerwania powodują VM exit
- które porty IO powodują VM exit

VM exits

Możliwe przyczyny, zależnie od konfiguracji:

- instrukcje uprzywilejowane
  - wszystko co związane ze stronicowaniem
  - instrukcje związane z wirtualizacją
- przerwania i ich konfiguracja (APIC, GIC)
- dostęp do przestrzeni IO
- *interrupt window exiting*
- instrukcje niekoniecznie uprzywilejowane
  - wszystko związane ze spaniem
  - rozpoznawanie modelu CPU
  - odczyt TSC
- hypercalls (ale nie `vmfunc`)

VM exits: przypadek szczególny

pause może zostać zostać użyte do detekcji spinlocków, w takim przypadku spinlocki już nie są *low latency*.

```
void spinlock::lock()
{
    while (_lock.exchange(true)) {
        while (_lock.load(std::memory_order_relaxed)) {
            asm volatile("pause");
        }
    }
}
```

x86: zagnieżdżone VM

...właściwie to namiastka wsparcia dla nich.

- VMCS które zawiera stan hosta, guesta, konfigurację VM nie ma jawnego formatu. Wszystkie modyfikacje są realizowane przez operacje `VMREAD` i `VMWRITE`.
- Jeśli guest marzy o zostaniu hostem VM exit przy każdym zapisie i odczycie do VMCS to zły początek.
- Rozwiązanie: w trybie VMX non-root wspomniane wyżej instrukcje operują na **Shadow VMCS** wskazanym przez obecną VMCS.

Stronicowanie w VM

Proces

virtual address $\longrightarrow$ physical address

VM dawniej

guest virtual address $\longrightarrow$ physical address

VM dziś

guest virtual address $\longrightarrow$ guest physical address $\longrightarrow$ physical address

Stronicowanie w VM i TLB

- nowe rodzaje wpisów w TLB: stage 1, stage 2, combined
- **Virtual Processor Identifier** - ASID dla VM
- usuwanie wpisów tylko z konkretnego VPID
- usuwanie wpisów tylko z stage 2

Przerwania i VM

- wyjątki – zwykle nie trzeba angażować hypervisora
- IRQ i IPI – wirtualizacja kontrolera przerwań (local APIC/GIC/sAPIC/...[1])
  - wirtualizacja odczytów i zapisów
  - Self-IPI proste do realizacji bez VM exit
  - dostarczanie przerwań (i EOI) bez VM exit
  - wysyłanie przerwań do vCPU bez VM exit – posted-interrupts

[1] niepotrzebne skreślić

x86: posted-interrupts

1. CPU0 chce wysłać przerwanie `0x123` do vCPU wykonywanego na CPU1
2. CPU0 ustawia bit `0x123` w *posted-interrupt requests*
3. CPU0 wysyła IPI do CPU1 o wektorze takim jak w *posted-interrupt notification vector*
4. CPU1 dostarcza przerwanie do vCPU bez wykonywania VM exit

IOMMU

Jak dać VM możliwość kontaktu ze światem i nie tylko?

- emulowanie prawdziwego sprzętu z prawdopodobnie dużą ilością VM exits
- zaimplementowanie nowego interfejsu specjlanie dla VM z mniejszą ilością VM exits (np. VirtIO)
- bezpośrednie udostępnienie urządzenia
  - co z brakiem zaufania dla VM?
  - VM ma inne pojęcie na temat adresów fizycznych niż urządzenie

IOMMU

- translacja adresów w DMA
- *interrupt remapping*
- *interrupt posting*

IOMMU – device address translation

- każde urządzenie jest przypisane do jakiejś domeny
- adres urządzenia w PCI: *bus* (8 bitów), *device* (5 bitów), *function* (3 bity)
- 2-poziomowa hierarchiczna translacja: *root table* zawiera wskaźniki do *context-table* dla każdej z szyn, *context-table* wpisy dla każdego urządzenia i funkcji na danej szynie
- wpisy w *context-table* to wskaźniki do tablicy PML4 tłumaczące adresy DMA
- skoro jest translacja adresów to jest także cache: Device-TLB
- żądania DMA mogą zawierać address space identifier, wtedy wszystko się bardziej komplikuje, ale to nie dotyczy VM

IOMMU – interrupt remapping

Interrupt Remapping Table – wpis *n* zawiera informację o tym co zrobić z przerwaniem o indeksie *n*, tj. albo określa procesor docelowy i wektor albo nakazuje wykonać *interrupt posting*.

Specyfikacja przewiduje istnienie pamięci podręcznej wpisów z IRT.

IOMMU – interrupt posting

IRTE zawiera wskaźnik do *Posted Interrupt Descriptor* z informacjami:

- wskaźnik do *posted-interrupt requests*
- *posted-interrupt notification vector*
- ID procesora

Dziękuję za uwagę.