معماری و چالش‌های توسعه هایپروایز

پژوهشگر: سعید شهبازی

استاد راهنما: میلاد کهساری الهادی

دوره اول مهندسی معکوس و تحلیل باینری[1]

تاریخ ارائه: 1-6-1401

[1] https://aiooo.ir

فهرست

مقدمه‌ای بر مجازی‌سازی 4

هایپروایزر و چالش طراحی آن برای پردازنده‌های x86 4

معماری x86 بدون زیر ساخت مجازی‌سازی 6

چالش‌های مجازی‌سازی در معماری x86 7

رویکرد 3/1/0 8

رویکرد 3/3/0 8

چالش اول - Ring Aliasing 9

چالش دوم - Address-space compression 9

چالش سوم - Nonfaulting access to privileged state 10

چالش چهارم - Adverse impacts on guest transitions 10

چالش پنجم - Interrupt virtualization 10

وقفه‌های قابل نادیده گرفته شدن 12

معماری VLAPIC 15

چالش ششم - Ring Compression 15

انواع تکنیک‌ها ترجمه درخواست‌های Guest OS 15

تکنیک ترجمه باینری 16

تکنیک فرامجازی‌سازی 18

مجازی‌سازی با کمک سخت افزار 19

مکانیزم Page-table Shadowing Logical Processors 21

فرامجازی‌سازی MMU 23

جدول صفحات تاریک - Shadow Page Tables 24

مکانیزم Hardware assisted paging 25

مکانیزمDMA Remap Physical Memory 27

منابع.. 27

مقدمه‌ای بر مجازی‌سازی

وقتی میگوییم آموزش مجازی منظور این نیست که حقیقتاً آموزش نیست بلکه منظور این است که آموزش در فضای حضور جسمانی معلم و متعلم صورت نمی‌گیرد اما درعین‌حال این آموزش واقعاً یک آموزش است. همین‌طور است اصطلاحاتی چون ماشین مجازی، دنیای مجازی، حافظه مجازی، و ... همه این اصطلاحات به اموری حقیقی اشاره دارند. با این حال، وقتی در مورد ماشین مجازی صحبت می‌کنیم، مجازی‌سازی به ساخت نمونهٔ غیر واقعی از چیزهایی مثل پلتفرم سخت‌افزاری، سیستم عامل، وسایل ذخیره‌سازی یا منابع شبکه، گفته می‌شود.

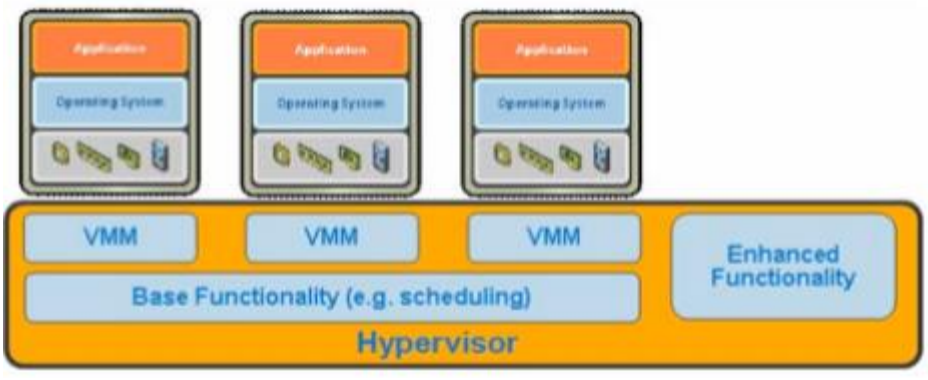
مجازی‌سازی از یک نوع تفکر عمیق و اجرا کردن هر آنچه که در فکر و ذهن می‌گذرد و نهایتاً بدون وجود خارجی پیاده‌سازی می‌گردد. در علم کامپیوتر استفاده از تکنولوژی مجازی‌سازی باعث رشد و پیشرفت بسیار شده است. پیاده‌سازی دستگاه‌های سخت‌افزاری به صورت مجازی اما با همان عملکرد مزایای بسیاری را برای ما به به رهاورد کشیده است.

اصولاً نرم‌فزارها مجازی هستند چون ذات آنها فیزیکی نیست. از اینرو می‌توان گفت مجازی‌سازی در اکثر اوقات شکل نرم‌افزاری دارد؛ که البته بر روی یک سخت‌افزار خاص اجرا خواهد شد. طراحی و شبیه‌سازی انواع سوییچ‌ها، روترها، سرورها و ... از این دسته‌اند. شرکت‌هایی نیز در زمینه تولید سیستم‌های مجازی مشغول به کارند نظیر شرکت مایکروسافت با سیستم Hyper-V یا سیستم‌های مبتنی بر هسته لینوکس از جمله ESXI با مجموعه نرم افزارهای vmware که هایپروایزرهایی به منظور ایجاد ماشین مجازی ارائه می‌دهند.

هایپروایزر و چالش طراحی آن برای پردازنده‌های x86

هایپروایز یا به عبارت دیگر ناظر ماشین مجازی (VMM)[1] یک لایه نرم‌افزاری در معماری Hosted و یا یک سیستم‌عامل در معماری Baremetal است. هایپروایزرهای خانواده Baremetal در قالب یک سیستم‌عامل به طور مستقیم بر روی سخت‌افزار نصب شده و تمامی ماشین‌های مجازی را مدیریت می‌کنند. مثلا هایپروایز XEN از معماری Baremetal و هایپروایزر VMware Workstation از معماری Hosted استفاده می‌کنند.

[1] Virtual Machine Monitor



تصویر 1: نحوه مدیریت ماشین‌های‌مجازی توسط هایپروایزر

در معماری VMware برای پردازنده‌های خانواده x86 برای هر ماشین‌مجازی یک VMM جهت ایجاد یک لایه انتزاعی[1] یا به زبان دیگر یک لایه‌مجازی بین ماشین مجازی و سخت‌افزار ایجاد می‌شود که وظیفه تقسیم و تخصیص منابع سخت‌افزاری به ماشین‌های مجازی را بر عهده دارد. به این رویکرد تقسیم و تخصیص منابع به ماشین های مجازی به عنوان تکنیک Resource Multiplexing شناخته می‌شود.

قبل از سال 2007، پردازنده‌های مبتنی بر معماری x86 قابلیت پشتیبانی از زیرساخت مجازی‌سازی را نداشتند و شرکت‌های نظیر VMware و Xen، در صدد پیدا کردن روش‌هایی برای مجازی‌سازی در این معماری بودند. شایان ذکر است، پیش از سال 2007 سیستم‌عامل‌هایی که برای پردازنده‌های خانواده x86 طراحی و ایجاد شده بودند، باید به صورت مستقیم روی سخت‌افزار اجرا می‌شدند تا بتوانند از سخت‌افزار به صورت کامل استفاده کنند.

پردازنده‌های مبتنی بر معماری x86 دارای چهار رینگ اصلی برای اجرا و راه‌اندازی سیستم‌عامل است. در این معماری، پس از راه‌اندازی و اجرای مولفه مُد مدیریت سیستم (SMM)[2]، فریمور BIOS یا UEFI بارگذاری و اجرا خواهد شد. سپس با توجه به اینکه از فریمور BIOS یا UEFI استفاده شود، مراحل بوت سیستم‌عامل شروع خواهد شد. شایان ذکر است، مراحل بارگذاری سامانه عامل با توجه به نوع فریمور متفاوت خواهد بود.

در معماری x86، سیستم‌عامل در Ring 0 شروع به فعالیت می‌کند و در نتیجه می‌تواند instructionهای خود را به طور مستقیم بر روی پردازنده اجرا کند و هیچ وقفه‌ای[3] صورت نگیرد. پس سیستم‌عامل مجازی (در معماری Hosted) یا خود سیستم‌عامل هایپروایزر (در معماری Baremetal) باید در رینگی از پردازنده اجرا شود که زیر رینگ

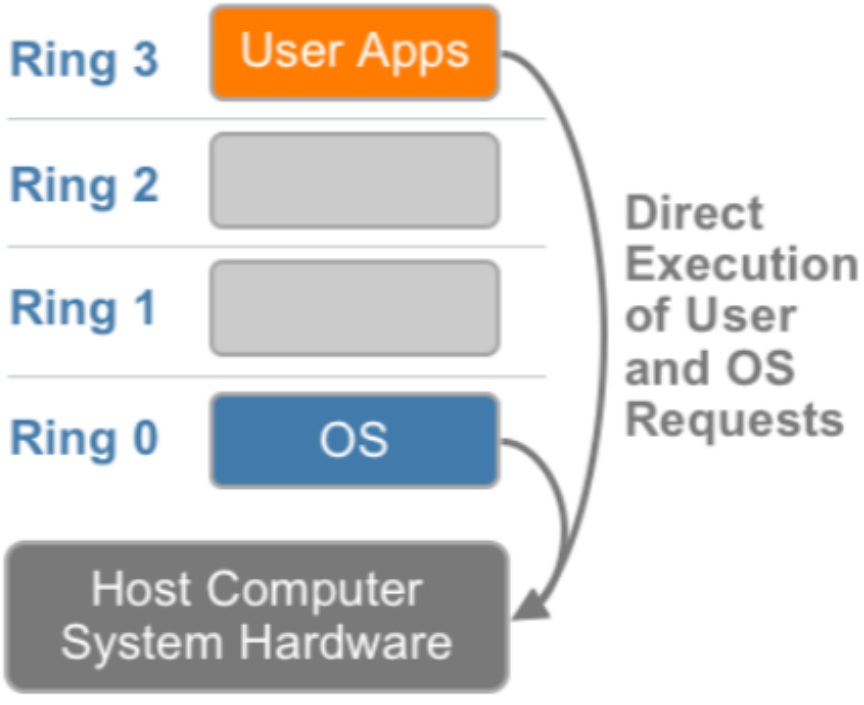
---

[1] Abstraction Layer

[2] System Management Mode

[3] Interrupt

سیستم‌عامل مجازی قرار بگیرد تا درخواست‌های سیستم‌عامل مهمان[1] از طریق این لایه به سمت سخت‌افزار ارسال گردد. در تصویر 2، مراحل گذر درخواست‌های برنامه‌های کاربردی و همچنین درخواست سرویس‌های خود ماشین‌مجازی نمایش داده شده است.



تصویر 1: مراحل ارسال درخواست‌های مجازی برنامه‌های کاربردی و سامانه‌عامل به سخت‌افزار

معماری **x86** بدون زیر ساخت مجازی‌سازی

چالش اصلی در مجازی‌سازی و مدیریت رفتار ماشین‌های‌مجازی در این است که سیستم‌عامل در Ringای اجرا می‌شود که کدهای اجرایی آن برای آن Ring نوشته نشده است و این باعث ایجاد خطا در اجرا و همچنین عملکرد سیستم‌عامل خواهد شد. از همین روی، در طراحی هایپروایزر باید روشی برای انتقال دستورالعمل‌های اجرایی سیستم‌عامل مهمان به سمت سخت‌افزار ایجاد شود.

برای رفع مشکل مذکور، توسعه‌دهندگان این حوزه مانند VMware از ساختاری به نام ترجمه باینری[2] و اجرای مستقیم[3] استفاده کردند و هایپروایزر خود را در لایه Ring 0 راه‌اندازی کردند که همین امر موجب شد سیستم عامل

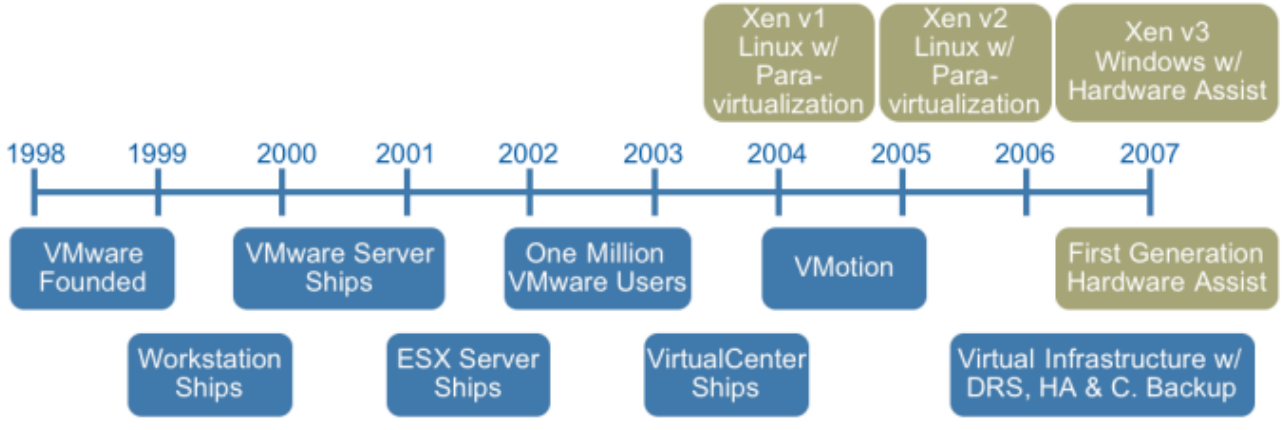
---

[1] Guest OS

[2] Binary Translation

[3] Direct Execution

مهمان در لایه سوم یا همان Ring 3 ولی با دسترسی بیشتر از برنامه های کاربردی و دسترسی کمتر از هایپروایزر اجرا شود. به این روش که توسط VMware ابداع شد، مجازی‌سازی کامل می‌گویند. در این روش، سیستم‌عامل مهمان هیچ اطلاعی از لایه دقیق اجرایی خود ندارد و تصور می‌کند که مالک سخت‌افزار است و می‌تواند به طور مستقیم با آن ارتباط برقرار کند. در تصویر 3، روند شکل‌گیری و توسعه VMware و همچنین XEN را مشاهده می‌کنید.



تصویر 3: شکل‌گیری و توسعه VMware و XEN

در تصویر 3، روند پیشرفت شرکت VMware را از زمان پایه گذاری این شرکت تا سالی که تولید پردازنده‌های سازگار با مجازی‌سازی شروع شد، مشاهده می‌کنید. شایان ذکر است، به صورت، در بحث مجازی‌سازی اکنون 3 تکنیک اصلی جهت کنترل دستورالعمل‌های سطح پایین وجود دارد. در بخش زیر، این سه تکنیک آورده شده است:

- Full Virtualization using Binary Translation
- Paravirtualization
- Hardware assisted virtualization

چالش‌های مجازی‌سازی در معماری x86

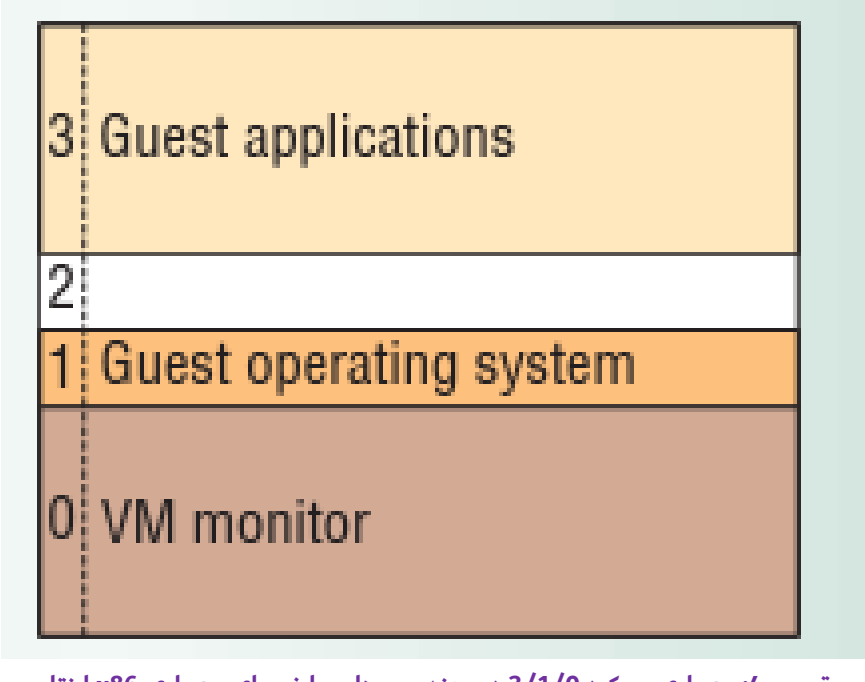
پردازنده‌های خانواده Intel در حالت کلی دو سطح دسترسی ارائه می‌دهند. سطح دسترسی 0 یا Ring 0 که همواره کرنل سامانه‌عامل در آن اجرا می‌شود، و سطح دسترسی 3 یا Ring 3 که برنامه‌های کاربردی در آن اجرا خواهند شد. سطوح دسترسی مشخص می‌کنند که دستورات سطح پایین که نیاز به دسترسی بالا دارند، در چه Ringای باید اجرا شوند، در غیر این صورت آن دستورات موجب ایجاد یک وقفه خواهند شد.

این سطوح دسترسی همچنین مشخص می‌کنند شما به چه بخشی از آدرس‌های حافظه می‌توانید دسترسی بگیرید. از همین روی، برنامه‌هایی که در سطح 3 در حال اجرا هستند، نمی‌توانند به داده‌هایی که در فضای آدرس کرنل هستند، دسترسی مستقیم بگیرند. به هر صورت، با توجه به این سطح‌بندی، سامانه‌عامل باید در سطح 0 اجرا شود که بتواند با پردازنده و سخت افزار به شکل صحیح تعامل و همچنین سخت‌افزار را کنترل کند.

از آنجایی که یک هایپروایزر نمی‌تواند چنین کنترلی را به ماشین مجازی بدهد، هایپروایزرهایی که برای معماری اینتل توسعه داده شده‌اند، از تکنیک Ring Deprivileging استفاده می‌کنند تا این مسئله را حل کنند. با استفاده از این تکنیک، تمامی نرم‌افزارهای سیستم‌عامل مهمان در یک سطحی بالاتر از سطح 0 اجرا خواهند شد. شایان ذکر است، دو تکنیک اصلی جهت شبیه‌سازی Ring 0 برای سیستم‌عامل مهمان وجود دارد:

رویکرد 3/1/0

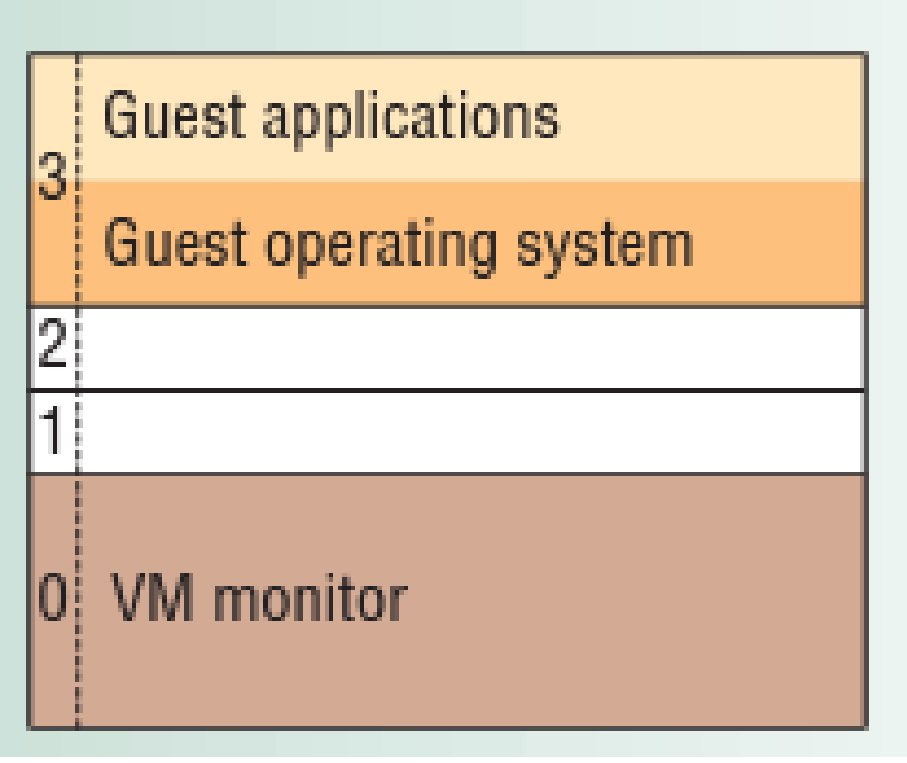
در این رویکرد، هایپروایزر یا VMM در Ring 0 اجرا می‌شود، سپس سیستم‌عامل مهمان در Ring 1 و برنامه‌های کاربردی درون سیستم‌عامل در Ring 3 اجرا خواهند شد. در تصویر 4، معماری این رویکرد نمایش داده شده است.



تصویر 4: معماری رویکرد 3/1/0 در مهندسی هایپروایزر برای معماری x86 اینتل

رویکرد 3/3/0

در این رویکرد، هایپروایزر یا VMM در Ring 0 اجرا می‌شود، سپس سیستم‌عامل مهمان و برنامه‌های کاربردی درون سیستم‌عامل مهمان در Ring 3 اجرا خواهند شد. در تصویر 5، معماری این رویکرد نمایش داده شده است.



تصویر 5: معماری رویکرد 3/3/0 در مهندسی هایپروایزر برای معماری x86 اینتل

در این‌جا، باید این نکته را ذکر کرد که بحث Deprivileging باعث ایجاد چالش‌هایی گردید چرا که در واقع با حضور هایپروایزر، سیستم‌عامل مهمان را مجبور به اجرا در Ring متفاوتی می‌کنیم. در حالیکه، سیستم عامل برای عملکرد صحیح خود نیازمند اجرا در سطح یا Ring 0 است. با این حال، بعد ابداع رویکرد Deprivileging، چالش‌ها را می‌توان به چند دسته زیر تقسیم‌بندی کرد:

چالش اول - Ring Aliasing

اولین چالش مربوط به زمانی است که نرم‌افزار یا همان سیستم‌عامل در سطح یا Ringای اجرا می‌شود که برای آن Ring نوشته نشده است و با اجرای دستورات سطح پایین، پاسخ دریافتی دارای خطا می‌باشد و سامانه برای ما یک وقفه ایجاد خواهد کرد که دستورات اجرایی نیاز به سطح دسترسی کرنل دارند از همین روی، عملکرد سیستم عامل و نرم افزارهای کاربردی دچار تداخل خواهد شد و سامانه پایداری خود را از دست می‌دهد.

چالش دوم - Address-space compression

بحث دیگر مربوط به محافظت از فضای حافظه است که هایپروایزر یا VMM در آن اجرا می‌شود. در واقع هایپروایزر در فضای آدرس‌مجازی سامانه‌عامل مهمان اجرا می‌شود تا بتواند به اطلاعات سامانه‌عامل مهمان دسترسی داشته باشد و با آن تعامل برقرار کند.

چالش سوم - Nonfaulting access to privileged state

در سیستم‌های فیزیکی وقتی سیستم‌عامل در Ring 0 اجرا می‌شود، به طبع به پردازنده و تمام فضای آدرس حافظه دسترسی مستقیم دارد و می‌تواند با اجرای دستورالعمل‌هایی مانند دستورالعمل‌های سطح پایین رجیسترهایی مانند LGDT، LIDT، LLDT، و همچنین TLR تعامل کند و همچنین نرم‌افزارها می‌توانند با خواندن محتوای رجیسترهایی مانند SGDT، SIDT، SLDT، و STR در هر Ringای پاسخ از سمت پردازنده دریافت کنند، ولی نکته مهم این است که این مجموعه دستورالعمل‌های سطح پایین وقتی با سطح دسترسی (یا Privilege) پایین‌تر از Ring 0 اجرا شوند، وقفه یا fault بر نمی‌گردانند و سامانه متوجه عدم اجرا و عملکرد صحیح این دستورات نخواهد شد. حال اگر در لایه هایپروایزر برای پاسخ‌دهی به این دستورالعمل‌ها آماده نباشد، یا نتواند این مجموعه دستورالعمل‌ها را شبیه‌سازی[1] کند، ماشین مجازی دچار مشکل و عدم پایداری خواهد شد.

چالش چهارم - Adverse impacts on guest transitions

Ring deprivileging با برخی امکاناتی که IA32 در اختیار می گذارد مطابقت ندارد. مانند دستورالعمل هایی نظیر SYSENTER و SYSEXIT که دستور العمل اول System callای است که باعث انتقال سریع User app Ring 3 به Ring 0 می شود و دستور دوم باعث برگشت به Ring 3 می شود. چالش مهم در این دستورات این است که وقتی Guest OS این Instruction setها را صدا بزند به جای اینکه Guest OS Kernel که Ring 0 است برود به VMM می رود و در نتیجه VMM باید هر دستور SYSENTER را برای Guest OS، شبه سازی یا Emulate کند. همچنین اجرای این دو دستور از طرف Guest OS به سمت VMM باعث ایجاد یک Fault می شود که همچنان VMM باید آن را Emulate کند.

چالش پنجم - Interrupt virtualization

قبل از پرداختن به بحث وقفه‌ها[2] در ساختار مجازی‌سازی، باید آن را در ساختار پردازنده و ماشین تشریح کنیم. وقفه را می‌توان مکانیزمی برای مدیریت اتفاقات سطح پایین سخت‌افزاری و نرم‌افزاری با الویت بسیار بالا توسط سیستم‌عامل دانست. به عبارت دیگر، وقفه یک سیگنال به پردازنده است که به پاسخگویی سریع آن نیاز است.

---

[1] Emulate

[2] Interrupt

هنگامی که یک وقفه رخ می‌دهد، پردازنده عملیات جاری خود را متوقف می‌کند تا به درخواست وقفه یا سیگنال دریافتی خود رسیدگی کند. پردازنده‌های خانوادهٔ x86 به وقفه‌های تولید شده به وسیلهٔ سخت‌افزار و نرم‌افزار پاسخ می‌دهند که به ترتیب به آن‌ها وقفه‌های سخت‌افزاری، و وقفه‌های نرم‌افزاری گفته می‌شود.

فرض کنید که قرار است بین یک دیوایس USB و پردازنده ارتباطی برقرار شود و هرگاه عملیات خواندن از روی USB تمام شد، پردازنده مطلع شود. اگر قرار بود یک بیت برای این کار در نظر بگیریم و پردازنده دائماً پایان عملیات خواندن را بررسی کند، سازوکاری هزینه برخواهد بود. اما نکتهٔ مهم اینجاست که وقفه نیاز به کنترل پردازنده ندارد و هر وقت عملیات خواندن تمام شود، کنترل‌کنندهٔ USB یک وقفه یا یک سیگنال برای پردازنده می‌فرستد که مشخص می کند، عملیات آن به پایان رسیده است.

به عبارت دیگر، وقفه‌ها راهی به منظور تعامل مستقیم با پردازنده برای رسیدگی به کارهای با الویت بالا هستند. به عنوان مثال وقتی یک دستگاه مبتنی بر USB را به کامپیوتر متصل می‌کنید، تمامی این رخدادهای سخت‌افزاری موجب ایجاد یک وقفه در سطح سیستم خواهد شد که در ادامه سامانه‌عامل متوجه این رخدادهای سخت‌افزاری می‌شود و در نهایت با فراخوانی روتین‌های سیستمی تعریف شده در سطح پردازنده آن رویداد سخت‌افزاری را پاسخ خواهد داد.

به عنوان مثال، وقتی موس را حرکت می دهید، این حرکت موجب ایجاد یک وقفه در سطح ماشین خواهد شد. این وقفه‌ها توسط سامانه‌عامل دریافت و به سمت پردازنده برای مدیریت و پاسخگویی ارسال می‌شوند. سامانه‌عامل وقتی متوجه چنین وقفه‌هایی می‌شود، فعالیت جاری پردازنده را متوقف می‌کند تا به وقفه پاسخ بدهد. حال بر اساس ارجحیت پروسه درخواست کننده و همچنین نوع وقفه، درخواست آن در صف قرار می‌گیرد تا پردازنده به آن پاسخ بدهد. به هر صورت، وقتی درباره IRQ در محیط ویندوز صحبت می‌کنیم، یعنی وقتی درخواست وقفه میاد، هر وقفه‌ای در یک سطح خاص باید پردازش شود که هر چی آن سطح بالاتر باشد، وقفه موازی با یک سطح پایین‌تر باید متوقف شود تا وقفه با سطح و الویت بالاتر تمام شود.

در سامانه‌عامل ویندوز، 32 سطح برای وقفه‌ها در معماری x86 و 16 سطح در معماری x64 داریم. در معماری x86، 3 سطح برای وقفه‌های نرم‌افزاری است، و مابقی متعلق به وقفه‌های سخت‌افزاری هستند و باید برای هر کدام یک ISR تعریف شود. آن سطوح نرم‌افزاری با عنوان PASSIVE_LEVEL و APC_LEVEL و DPC_LEVEL شناخته می‌شوند. در جدول 1، اطلاعات برخی از این سطوح دسترسی برای وقفه‌ها مبتنی بر هر معماری آورده شده است:

IRQL	Processor IRQL value			Description
	x86	x64	Itanium	
PASSIVE_LEVEL	0	0	0	User threads and most kernel-mode operations
APC_LEVEL	1	1	1	Asynchronous procedure calls and page faults

DISPATCH_LEVEL	2	2	2	Thread scheduler and DPCs
CMC_LEVEL	N/A	N/A	3	Correctable machine-check level (Itanium platforms only)
DIRQL[1]	3–26	3–11	4–11	Device interrupts
PC_LEVEL	N/A	N/A	12	Performance counter (Itanium platforms only)
PROFILE_LEVEL	27	15	15	Profiling timer for releases earlier than Windows 2000
SYNCH_LEVEL	27	13	13	Synchronization of code and instruction streams across processors
CLOCK_LEVEL	N/A	13	13	Clock timer
CLOCK2_LEVEL	28	N/A	N/A	Clock timer for x86 hardware
IPI_LEVEL	29	14	14	Interprocessor interrupt for enforcing cache consistency
POWER_LEVEL	30	15	14	Power failure
HIGH_LEVEL	31	15	15	Machine checks and catastrophic errors; profiling timer for Windows XP and later releases

جدول 1: سطوح یا الویت وقفه‌ها در ویندوز

در هر صورت، وقتی پردازنده در حال اجرا در یک IRQL با سطحی بالاتر از PASSIVE_LEVEL باشد، فعالیت‌های دیگر برای اینکه توسط پردازنده اجرا شوند، باید IRQL با سطحی بالاتر از PASSIVE_LEVEL داشته باشند. پردازنده با این رویکرد درخواست‌ها با الویت‌های مهم را مدیریت می‌کند. به هر صورت، در جدول 1، برخی از این سطوح مرتبط با وقفه‌ها در سیستم عامل ویندوز نمایش داده شده است.

همانطور که پیش از این ذکر شد، به غیر از سه سطح PASSIVE_LEVEL و APC_LEVEL و DPC_LEVEL، مابقی IRQLها معمولا در ارتباط با سخت‌افزارها باید تعریف شوند. بین این 29 سطح وقفه، یک سری سطح خاص دیگر هم وجود دارد البته که با سخت‌افزار خاصی متناظر نیستند و معمولا سطح خیلی بالایی برای کنترل وقایع توی خود سیستم، مثل Shutdown یا Dispatching یا ... دارند.

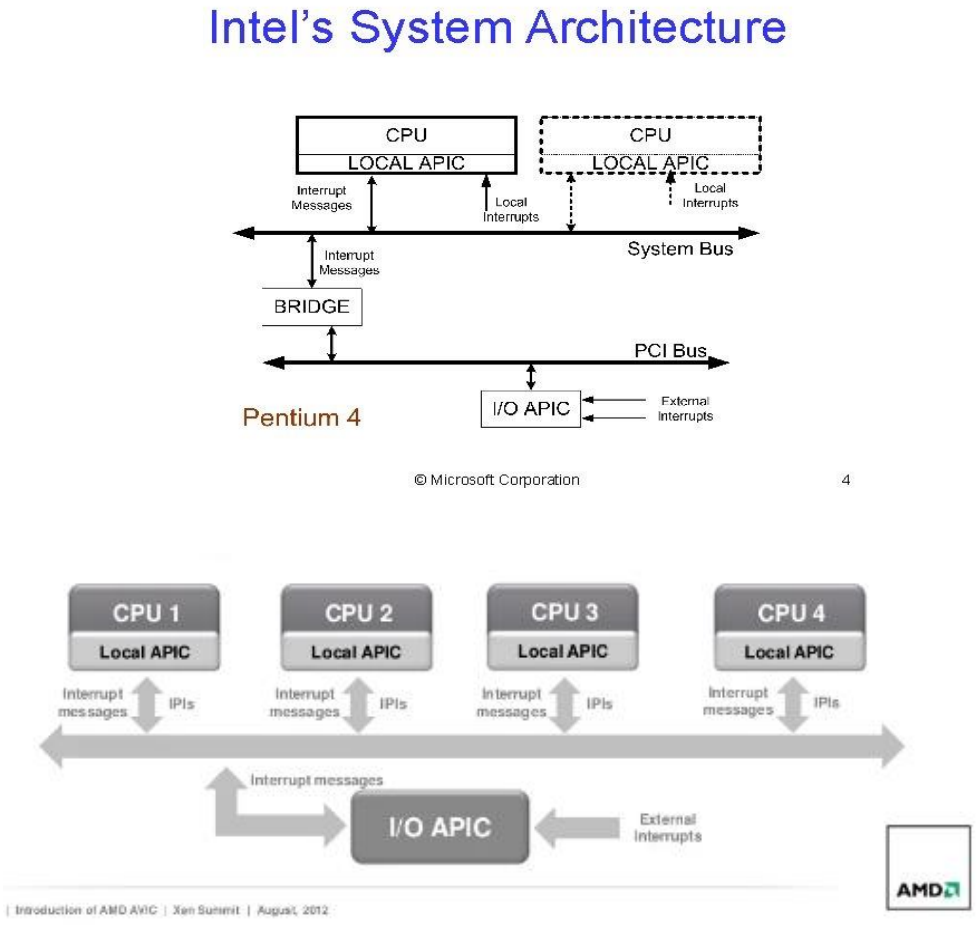
وقفه‌های قابل نادیده گرفته شدن[2]

پاسخگویی و کنترل وقفه‌های سخت‌افزاری را با دو تکنیک می‌توان مدیریت کرد. وقفه‌های سخت‌افزاری یا با تکنیک Maskable یا تکنیک Non-maskable مدیریت می‌شوند. وقفه‌ای که Maskable است، یعنی امکان نادیده گرفتن یا غیرفعال کردن آن توسط دستورالعمل‌های اسمبلی پردازنده وجود دارد. شایان ذکر است، این تکنیک در مدیریت وقفه‌ها برای کنترل اتصال Deviceها به کامپیوتر استفاده می‌شوند. وقفه‌های نوع Non-maskable امکان نادیده گرفته شدن یا غیرفعال شدن توسط دستورالعمل‌های اسمبلی پردازنده را ندارند. این نوع وقفه‌ها باید با سرعت بالایی پاسخ‌دهی شوند. به هر صورت، وقتی در مورد ماشین‌های مجازی و پیاده‌سازی هایپروایزرها صحبت می‌کنیم، در

[1] Device interrupt levels

[2] Maskable

حقیقت وظیفه اصلی مجازی‌سازی وقفه‌ها[1]، شبیه‌سازی و پیاده‌سازی معماری وقفه‌ها برای پاسخ‌دهی به رویدادهای ایجاد شده توسط پروسه‌های خارجی است. با این حال، قبل از اینکه وارد بحث مجازی‌سازی وقفه‌ها بشویم باید این مورد را در پلتفرم فیزیکی درک کنیم. در تصویر 6 معماری پردازنده اینتل نمایش داده شده است:

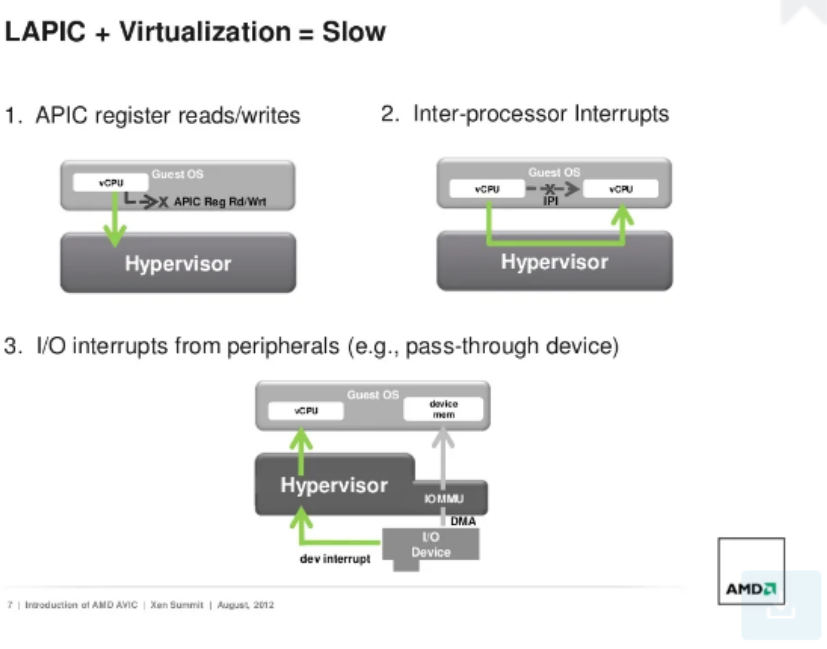


تصویر 6: معماری ایجاد و پاسخ‌دهی و اجرای یک وقفه خارجی

طبق شکل بالا وقتی یک وقفه خارجی توسط یک پروسه ایجاد می‌شود، به عنوان مثال وصل شدن یک دیسک خارجی به سیستم مراحل زیر صورت می‌گیرد تا پردازنده به وقفه ایجاد شده توسط یک دیوایس خارجی پاسخ دهد. I/O Device که در این مثال یک Flash Disk است یک درخواست وقفه از نوع I/O APIC به سمت سیستم عامل ارسال می‌کند. درخواست وقفه از طریق PCI Bus به System Bus ارسال می شود. در نهایت Local APIC موجود در پردازنده این وقفه خارجی را دریافت کرده و پردازنده شروع به اجرای آن می کند. در زیرساخت مجازی سازی و

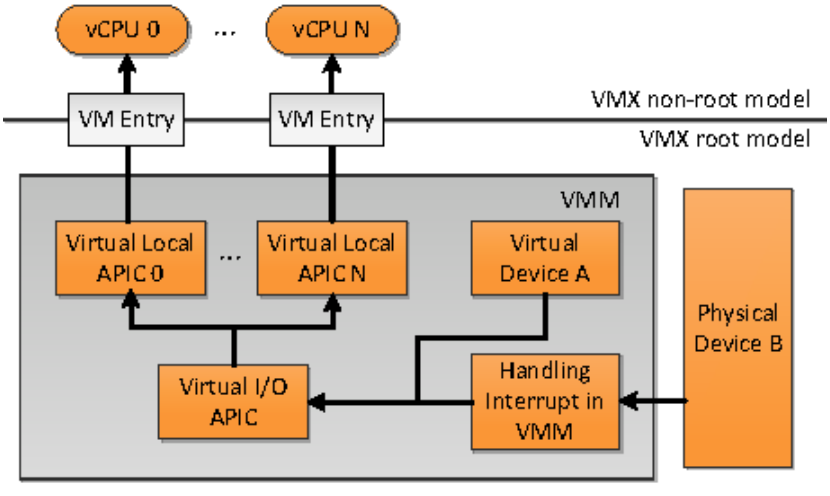
[1] Interrupt Virtualization

طراحی هایپروایزر، VMM همچنین باید یک رفتاری شبیه به محیط واقعی به منظور مدیریت و کنترل وقفه‌های داخلی و خارجی را شبیه‌سازی کند که مبحث Virtual Interrupt Architecture در طراحی هایپروایزرها به این مسئله می‌پردازد.



تصویر 7: معماری VLAPIC

مانند زیرساخت فیزیکی در زیر ساخت مجازی هر VCPU دارای یک Virtual Local APIC برای دریافت وقفه‌ها می‌باشد. همچنین Virtual I/O APIC یا Virtual PIC وقفه‌ها را در نهایت به سمت Local APIC ارسال می کند. در حقیقت تمامی این موارد، مولفه‌های نرم‌افزاری هستند که توسط VMM برای یک ماشین‌مجازی پیاده‌سازی می‌شود. مراحل این مدل به صورت زیر است:



تصویر 8: مُد Root و Nonroot

معماری VLAPIC

وقتی یک دستگاه مجازی نیاز به ارسال وقفه دارد، در مرحله اول اینترفیس Virtual I/O APIC را برای ارسال وقفه صدا می زند. Virtual I/O APIC یک Virtual Local APIC متناظر را برای ارسال وقفه انتخاب می‌کند. Virtual Local APIC از مکانیزم Intel VT-x برای تزریق وقفه ارسال شده به VCPU استفاده می‌کند. در معماری IA32 از رجیستر EFLAGS در پردازنده برای کنترل interrupt masking استفاده می‌شود. در بحث deprivileging، سیستم‌عامل مهمان با توجه به اینکه در Ring 0 قرار ندارد، اگر درخواست interrupt masking بدهد با fault مواجه می شود. برخی سیستم‌عامل‌ها به طور مداوم وقفه‌ها را mask و unmask می‌کنند و در حقیقت VMM این موارد را دریافت و شبیه‌سازی می کند و همین امر باعث افزایش overhead در زیرساخت Hypervisor می گردد.

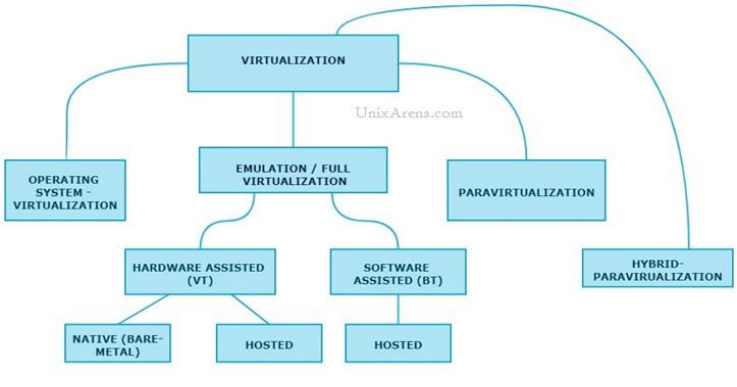
چالش ششم - Ring Compression

با استفاده از تکنیک Ring deprivileging، هایپروایزر در Ring متفاوت با سیستم‌عامل مهمان اجرا می‌شود و خود سیستم‌عامل مهمان در Ring 3 اجرا می‌شود. همچنین برنامه‌های کاربردی هم در Ring 3 اجرا خواهند شد. در نتیجه مشکلی که به وجود می‌آید دسترسی برنامه‌های کاربردی به سیستم‌عامل مهمان بواسطه CPL یکسان ایجا می‌شود، چون در یک رینگ عملیاتی یکسان حضور دارند.

انواع تکنیک‌ها ترجمه درخواست‌های Guest OS

توسعه دهندگان Hypervisorها برای رفع مشکلات مذکور سه راهکار زیر را ایجاد کردند.

1. Full Virtualization using binary translation OR Software Assisted Full Virtualization
2. Paravirtualization
3. Hardware assisted virtualization



تصویر 8: انواع معماری مجازی‌سازی و طراحی هایپروایزر

تکنیک ترجمه باینری[1]

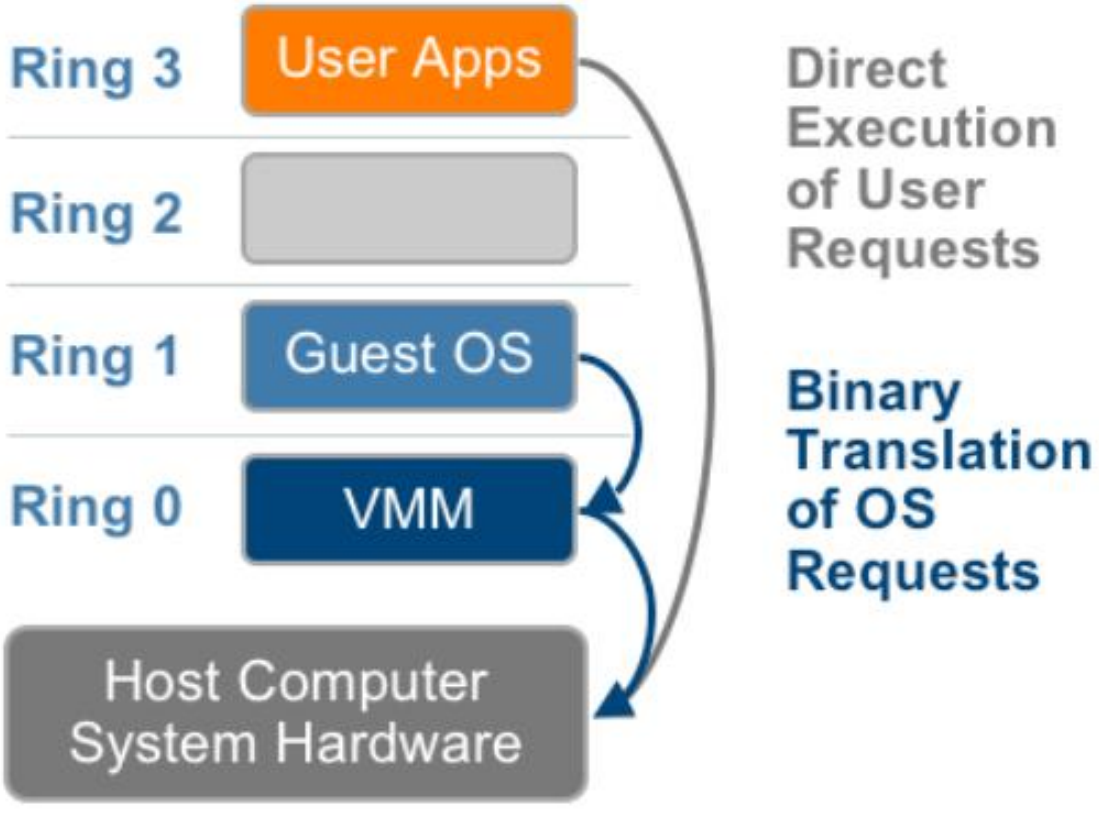
ترجمه باینری در اصل یک تکنیک مجازی‌سازی سیستم است. دستورالعمل‌های اجرایی حساس در باینری سیستم‌عامل مهمان یا با فراخوانی‌هایی از نوع Hypercall جایگزین خواهند شد که به صورت ایمن می‌توانند این دستورات حساس را کنترل کنند یا با opcodeهایی جایگزین خواهند شد که موجب یک Trap خواهند شد و در نهایت هایپروایزر می‌تواند متوجه Trap شود و آن را کنترل کند.

در اکثریت پردازنده‌های مدرن، دستورالعمل‌های تغییر زمینه یا همان Context Instruction قابل مجازی‌سازی نیستند. ترجمه باینری راه حلی به منظور حل این مشکل است. به عنوان مثال، اگر سیستم‌عامل مهمان بخواهد وضعیت پردازنده CPU را که حاوی پرچم‌ها/فیلدهای بیتی مهم است، تغییر دهد/بخواند، برنامه میزبان باینری سیستم‌عامل مهمان را برای چنین دستورالعمل‌هایی اسکن می‌کند و آن‌ها را با فراخوانی به هایپروایزر یا کدهای بازخوانی ساختگی جایگزین می‌کند. از طرف دیگر، Para-Virtualization تکنیکی دیگری است که در آن کد منبع سیستم‌عامل مهمان اصلاح می‌شود. در این تکنیک، تمام کدهای مربوط به دسترسی به منابع سیستم با APIهای Hypervisor اصلاح خواهند شد تا مشکل دستورالعمل‌های Non-Virtualizable حل شود.

به هر صورت، در رویکرد Full Virtualization از ویژگی ترجمه باینری یا Binary Translation برای به دام انداختن و همچنین مجازی‌سازی دستورات Non-Virtualizable که قرار است از سطح سیستم‌عامل مهمان به سمت سخت‌افزار واقعی ارسال شود، توسط سیستم‌عامل یا هایپروایزر در Ring 0 به دام انداخته می‌شود. سپس در یک ترتیب و توالی جدید دستورات سیستم عامل مهمان توسط VMM ترجمه می‌شود چرا که برخی از instructionها نیاز مند اجرا در Ring 0 هستند.

نکته دیگر این است که برنامه های کاربردی که عموما در Ring 3 اجرا می شوند به دلیل افزایش سرعت به طور مستقم Instruction های خود را به سخت افزار ارسال می کنند، در حقیقت در Full Virtualization سیستم عامل مهمان یک Hardware call ایجاد می کند. در تصویر 8، این موضوع نمایش داده شده است.

[1] Binary Translation



تصویر 8: استفاده از ویژگی ترجمه دودویی در معماری X86

تنها روشی که پیش نیاز سخت‌افزاری و تغییرات در سیستم‌عاملی ندارد، روش ترجمه باینری یا ترجمه دودویی است. از ویژگی‌های این تکنیک در مجازی‌سازی می‌توان به موارد زیر اشاره کرد:

1. سیستم‌عامل مهمان هیچ ایده‌ای از این که در یک محیط مجازی است ندارد
2. هایپروایزر تمام دستورالعمل‌های سیستم‌عامل مهمان را به صورت on the fly ترجمه و همچنین برای افزایش کارایی و استفاده مجدد در آینده cache می‌کند در حالی که دستورالعمل‌های برنامه‌های کاربردی در Ring 3 بدون کاهش سرعت به سخت افزار ارسال می‌شوند.
3. به دلیل استفاده از قابلیت ترجمه باینری یا ترجمه دودویی کارایی و سرعت سیستم عامل مهمان کمتر از سیستم عامل میزبان می باشد.
4. استفاده از قابلیت کش ترجمه دودویی باعث استفاده هر چه بیشتر Memory در سیستم عامل میزبان می‌شود.
5. کارایی سیستم‌های مبتنی بر ساختار مجازی‌سازی X86 همراه با ترجمه دودویی بین 80% تا 96% سیستم‌عامل میزبان می‌باشد.

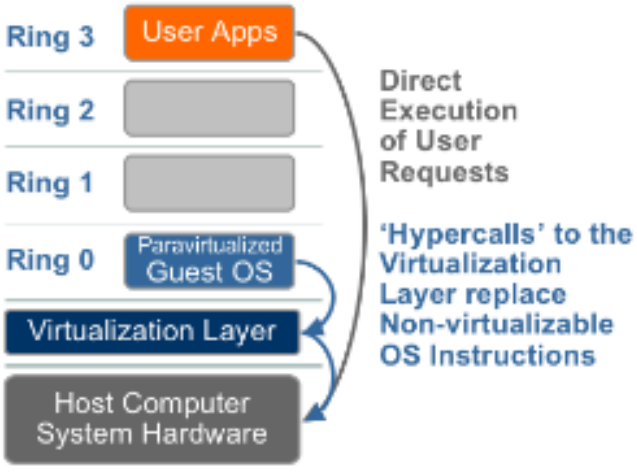
تکنیک فرامجازی‌سازی[1]

نام دیگر این تکنیک OS Assisted Virtualization می‌باشد و دلیل نام‌گذاری آن هم به خاطر دخیل بودن سیستم‌عامل مهمان در پروسه استفاده از این زیرساخت است. در حقیقت سیستم‌عامل مهمان برای سازگاری با محیط مجازی‌سازی تغییر داده می شود و در نهایت مجدد کامپایل خواهد شد. از همین روی، به این تکنیک فرامجازی‌سازی گفته می‌شود.

تغییر دادن سیستم‌عامل مهمان درکرنل‌های متن باز از جمله کرنل سامانه‌عامل لینوکس کار راحتی است چرا که کرنل سیستم‌عامل باز یا اصطلاحا Open-Source است و قابلیت تزریق درایور و همچنین اعمال تغییر در کرنل آن وجود دارد. ولی برخی سیستم‌عامل‌ها مانند ویندوز که اصطلاحا متن بسته یا Closed-Source هستند، فقط امکان نصب یکسری درایورها مانند VMware Tools بر روی آن ها وجود دارد و ما امکان اعمال تغییر بر روی خود کرنل را نداریم. به هر صورت، همانطور که پیش از این ذکر شد، در این تکنیک، هدف جایگزینی دستورالعمل‌های Non-Virtualizable با Hypercallها یا همان فراخوانی‌های هایپروایزر است تا سربار شبیه‌سازی از بین برود و سیستم‌عامل مستقیم به جای اینکه سعی در ارتباط با سخت افزار اصلی را داشته باشد، با Hypervisor مستقیما صحبت کند. شایان ذکر است، در تکنیک فرامجازی‌سازی سیستم‌عامل مهمان از اینکه در یک محیط مجازی است اطلاع دارد.

ایجاد یک ساختار ترجمه دودویی برای دستوراالعمل‌های سیستم‌عامل مهمان در تکنیک Full Virtualization کاری پیچیده و زمان بر است، ولی ایجاد یک سامانه‌عامل مهمان دستکاری شده به مراتب راحت می‌باشد برای همین شرکت‌هایی مانند VMware سال‌هاست که با ابزارهایی نظیر VMware Tools سعی در پیاده‌سازی تکنیک Paravirtualization دارد. در حقیقت ابزاری مانند VMware Tools جهت ایجاد یک درپشتی (Backdoor) به سمت هایپروایزر می‌باشد تا مواردی نظیر Graceful shutdown یا جهت نصب درایور کارت شبکه VMXnet را پوشش دهد. در حقیقت در این تکنیک سیستم عامل مهمان به طور مستقیم به سیستم عامل میزبان که در اینجا منظور Hypervisor است، از طریق درایورها دسترسی پیدا می‌کند.

[1] Paravirtualization

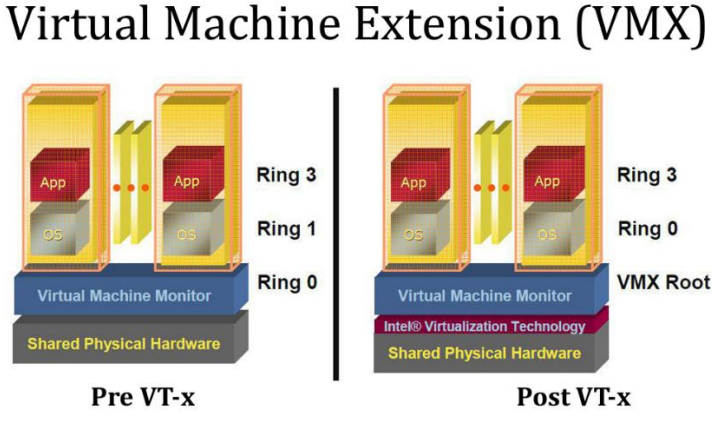


تصویر 9: معماری فرامجازی‌سازی

از ویژگی‌های استفاده از این تکنیک در مجازی‌سازی می‌توان به افزایش کارایی و سرعت نزدیک به محیط فیزیکی به واسطه کاهش شدید سربار در زیر ساخت مجازی‌سازی اشاره کرد. شایان ذکر است، به دلیل تغییرات زیاد در کرنل سیستم‌عامل مهمان این نوع مجازی‌سازی نیازمند پشتیبانی می‌باشد.

مجازی‌سازی با کمک سخت افزار

تکنیک Hypercalls در سال 2006 توسط دو تولید کننده پردازنده معرفی شد که شرکت Intel با تکنولوژی VT-x و شرکت AMD با تکولوژی AMD-V اجازه دادند VMMها در Mode جدیدی به نام Root Mode اجرا شوند که در واقع زیر Ring 0 می‌باشد و فراخوانی‌های سیستمی و دستورالعمل‌های حساس سیستم‌عامل مهمان به طور اتوماتیک بدون استفاده از تکنیک‌های Paravirtulization و Binary translation در VMM به دام می‌افتند.



تصویر 10: معماری VT-x

در این تکنیک دو نوع انتقال وجود دارد:

1. VM entry که مربوط به انتقال از VMX Root به سمت VMX non Root است.
2. VM exit مربوط به انتقال از VMX non Root به VMX Root است.

جدول زیر مقایسه سه تکنیک مذکور است:

	Full Virtualization with Binary Translation	Hardware-Assisted Virtualization	OS-Assisted Virtualization and Paravirtualization
Technique	Binary Translation and Direct Execution	Exit to Root Mode on Privileged Instructions	Hypercalls
Guest Modification and Compatibility	Unmodified Guest OS and Excellent compatibility	Unmodified Guest OS and Excellent compatibility	Guest OS codified to issue Hypercalls, so it cannot run on Native Hardware or other Hypervisors Poor compatibility; Not available on Windows OSes
Performance	Good	Fair Current performance lags Binary Translation virtualization on various workloads but improves over time	Better in certain cases
Used By	VMware, Microsoft, Parallels	VMware, Microsoft, Parallels, Xen	VMware, Xen
Guest OS Hypervisor Independent?	Yes	Yes	XenLinux runs only on Xen Hypervisor VMI-Linux is Hypervisor agnostic

تصویر 11: مقایسه سه تکنیک مذکور

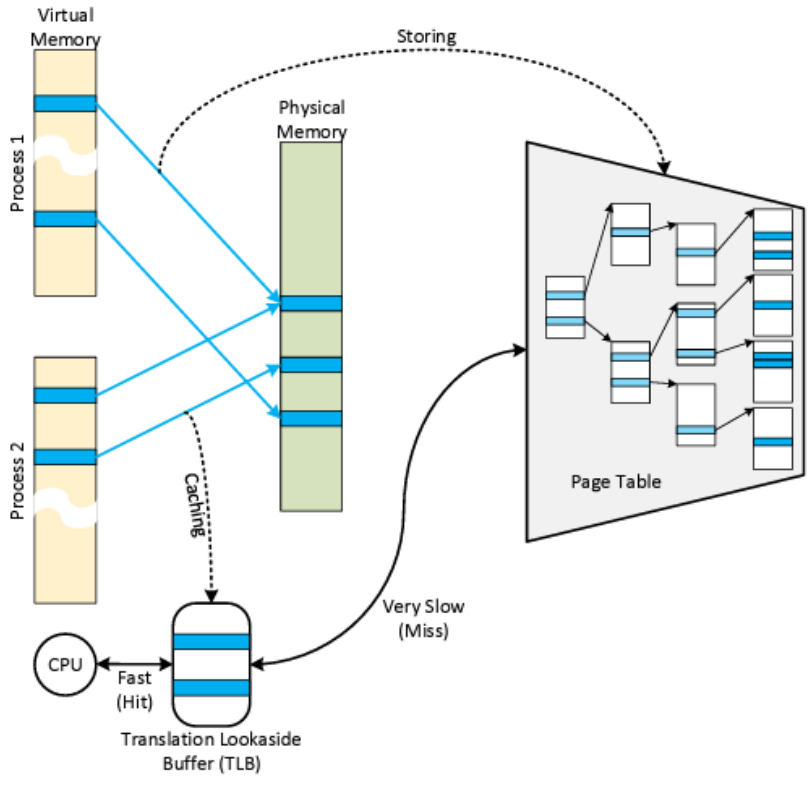
مکانیزم Page-table Shadowing Logical Processors

هر پروسه در سیستم‌عامل ویندوز دارای حافظه مجازی برای ذخیره داده در فضای آدرس مختص خود است. به عنوان مثال نرم افزار فایرفاکس دارای حافظه مجازی خود است. حافظه مجازی قسمتی جدای ناپذیر از سیستم‌عامل ویندوز به شمار می‌رود. هر صفحه مجازی یا Virtual Page (صفحات موجود در حافظه مجازی) در سیستم عامل به یک آدرس فیزیکی اشاره می‌کند در نتیجه تمام اطلاعات مرتبط با ترجمه آدرس‌های مجازی به فیزیکی در جایی به نام Page table ذخیره می‌شوند.

پردازده وظیفه ترجمه آدرس های مجازی به آدرس های فیزیکی را بر عهده دارد. شایان ذکر است، به دلیل افزایش Performance یا سرعت اجرا ماشین آدرس‌های ترجمه شده یا به بیان دیگر Page table entriesها پر استفاده در فضایی به نام TLB[1] ذخیره می‌شوند تا در صورت درخواست سیستم‌عامل مبنی بر صدا کردن پارامتر یا متغیری، پردازنده محل آن را از حافظه اصلی مجدد بازیابی نکند و از حافظه کش خود که TLB نام دارد، به منظور دسترسی به آن اطلاعات استفاده کند. پس وظیفه TLB افزایش سرعت ترجمه بین آدرس مجازی و آدرس فیزیکی است که این روش با ذخیره page-tableها در واحدی بین پردازنده و حافظه اصلی ماشین انجام می شود. پس روش پیدا کردن یک Physical memory page بدین صورت است که در مرحله اول پروسه درون فضای آدرس‌دهی خود به دنبال یک PTE می‌گردد و قبل از اینکه در خواست به حافظه فیزیکی یا حافظه اصلی برسد، بین راه توسط پردازنده بررسی می‌گردد که آیا در بافر TLB قرار دارد یا خیر و اگر وجود نداشت TLB Miss شکل می‌گیرد.

شایان ذکر است، در زمانی که سیستم‌عامل درخواستی را به پردازنده جهت پردازش ارسال می‌کند و پردازنده متوجه عدم حضور PTEهای مورد نیاز در TLB می‌شود، باید به روشی PTE را بدست بیاورد که دو راه وجود دارد. روش کار TLB به این صورت است که وقتی نیاز به ترجمه آدرس مجازی به آدرس فیزیکی است، در مرحله اول پردازنده حافظه کش خود که TLB نام دارد را چک میکند و اگر در Cache موجود نبود یک وقفه ایجاد شده و سیستم عامل برای پیدا کردن page table اقدام می‌کند. اگر سیستم عامل توانست در حافظه کش خود این آدرس مپ شدهپ را پیدا کند آدرس آن را در TLB ذخیره خواهد کرد.

[1] Translation lookaside buffer



تصویر 12: نحوه واکشی اطلاعات مموری و نقش TLB

روش اول وابسته به سیستم عامل است زیرا باعث می‌شود که پردازنده یک trap سمت سیستم عامل جهت به دست آوردن PTE و درج آن در TLB ایجاد کند که به واسطه راهکار نرم افزاری دارای Latency زیادی است.

روش دوم که روش پراستفاده‌تری است راهکار سخت‌افزاری می‌باشد که دیگر نیاز به ارسال trap به سمت سیستم‌عامل نیست. در این روش پس از TLB miss پردازنده پروسه‌ای به نام Page Table Walker را اجرا می‌کند که در فضای حافظه فیزیکی به دنبال PTE مورد نظر بگردد و آدرس آن را در TLB ذخیره کند.

Year	Processor	L1 TLB size	L2 TLB size
1999	Pentium III	72	0
2004	Pentium 4	64	0
2008	Nehalem	96	512
2012	Ivybridge	100	512
2014	Haswell	100	1024
2015	Skylake	100	1552

تصویر 13: سایز TLB در پردازنده های قدیمی

در بحث مجازی‌سازی به دلیل وجود دو سیستم‌عامل که هر دو نیازمند اجرای در Ring 0 هستند، ولی در واقع فقط یکی از آن‌ها این قابلیت را دارد، معماری متفاوت می‌شود. در معماری مجازی‌سازی دیگر واژه تبدیل آدرس مجازی به آدرس فیزیکی یا V2P مطرح نیست چرا که دو واژه gVA[1] و gPA[2] مطرح می‌شوند که باید به یکدیگر Map شوند. ایت اتفاق فقط در سطح سیستم عامل مهمان است و سپس در لایه بعدی gPA باید به hPA[3] ترسیم یا Map گردد و همین امر باعث افزایش Latency در مواقع TLB miss خواهد شد پس در حقیقت در ساختار مجازی‌سازی یک لایه دیگر نیز اضافه می‌شود. در تصویر 14، این مورد نمایش داده شده است.

تصویر 14: مراحل تبدیل آدرس های مجازی به فیزیکی در ماشین مجازی

VMM وظیفه کنترل p2m و دریافت اطلاعات mapping برای v2p را بر عهده دارد. به هر صورت، سه روش در ساختار مجازی‌سازی در بحث Memory virtualization وجود دارد:

فرامجازی‌سازی MMU[4]

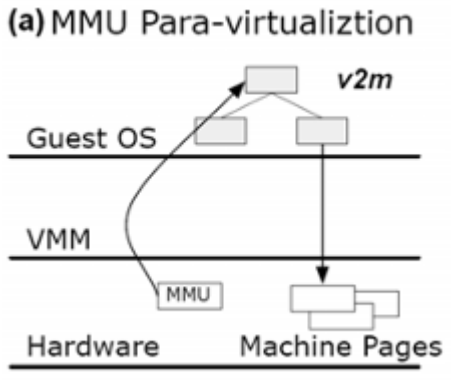
در این تکنیک با دستکاری کرنل سیستم‌عامل مهمان، لایه هایپروایزر حذف می‌شود و Mapping بین Virtual2Physical توسط سیستم‌عامل مهمان به طریقی انجام می‌شود که گویا در Ring 0 قرار دارد و مستقیم توسط Memory management Unit مدیریت می‌شود. در تصویر 15، معماری این رویکرد نمایش داده شده است.

[1] Guest Virtual address

[2] Guest physical address

[3] Host Physical address

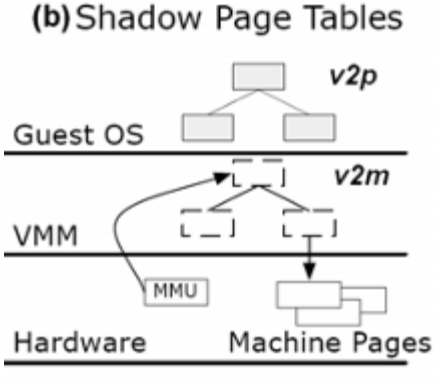
[4] MMU Paravirtualization

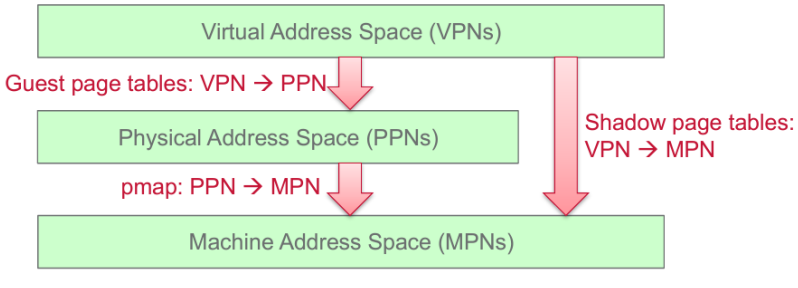


تصویر 15: معماری مجازی سازی Memory در ساختار Paravirtualization

جدول صفحات تاریک - Shadow Page Tables

در این راهکار سیستم‌عامل مهمان V2P خودش را دارد و یک page table دیگر به نام shadow page table توسط VMM نگهداری می‌شود که وظیفه ترسیم آدرس ها بین سیستم‌عامل مهمان و آدرس های میزبان را بر عهده دارد. در این روش VMM وظیفه ایجاد ارتباط بین جدول صفحات تاریک با MMU را بر عهده دارد و همچنین باید تمام تغییرات در جدول صفحه مهمان در Shadow Page ایجاد شود و چون سیستم‌عامل مهمان هیچ اطلاعی از VMM ندارد، این مورد وظیفه VMM است تا فضای آدرس دهی ماشین مهمان را مانیتور کند.

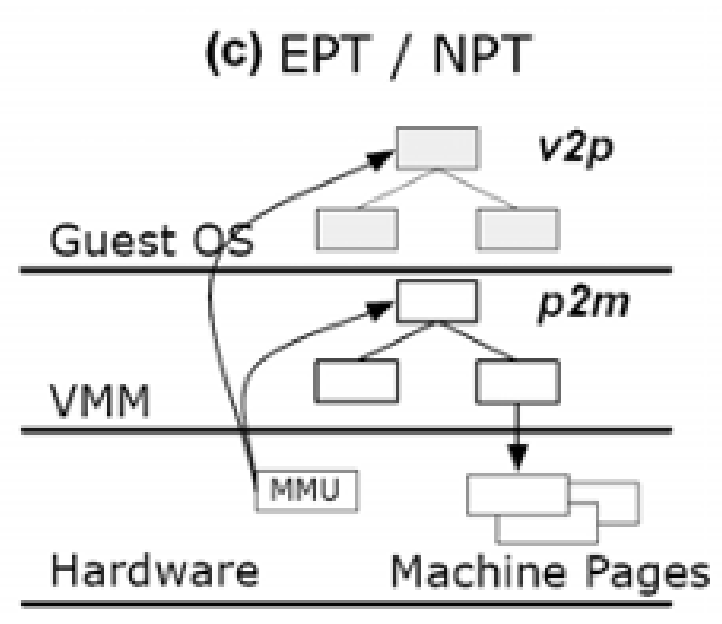




تصویر 16: معماری مجازی‌سازی حافظه در ساختار Full virtualization with Software Assist

مکانیزم Hardware assisted paging

شرکت Intel و AMD به ترتیب از قابلیت EPT (extended page table) و NTP (nested page table) استفاده کردند. در این روش MMU یک جدولی برای ترجمه آدرس‌های مجازی مهمان به آدرس‌های فیزیکی مهمان دارد و همچنین یک extended page هم برای ترجمه آدرس‌های فیزیکی مهمان به آدرس‌های فیزیکی میزبان دارد. پس با HAP یک ترجمه ادرس برای اینکه به سرانجام برسد باید از دو جدول Guest table و Extended table عبور کند.



تصویر 17: معماری مجازی‌سازی حافظه در ساختار hardware assist

طبق بررسی های انجام شده در سال 2006 که این تکنولوژی معرفی شده بود، تفاوت کارایی چندانی بین Shadowing و HAP وجود نداشت. در تصویر 18، نتیجه این مقایسه نمایش داده شده است.

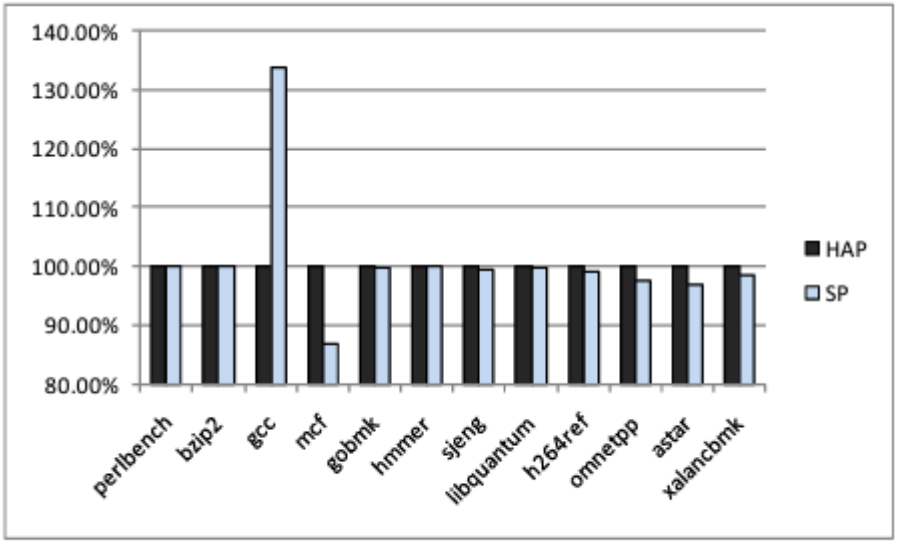


Figure 3. Normalized execution time (SPEC Int).

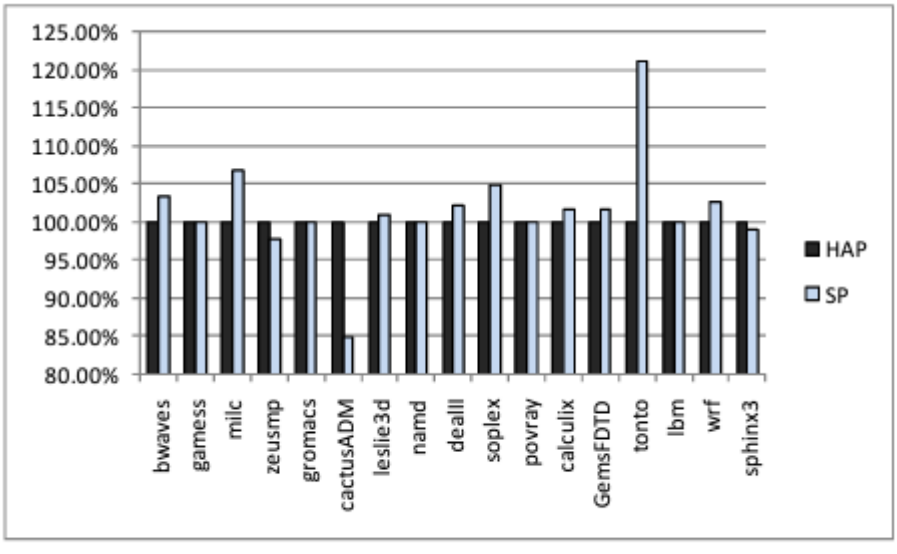


Figure 4. Normalized execution time (SPEC FP)

تصویر 18: مقایسه دو تکنیک Shadow paging و HAP

مکانیزم DMA Remap Physical Memory

مکانیزم Direct memory access قابلیتی است که در پردازنده‌های اینتل جهت انتقال سریع‌تر اطلاعات استفاده می‌شود. در حقیقت این قابلیت این اجازه را به سیستم می‌دهد تا Deviceها به طور مستقیم و بدون دخالت پردازنده با RAM، اطلاعات رد و بدل کنند.

در نبود این تکنولوژی وقتی Deviceای بخواهد از فضای Memory استفاده کند تا پایان اتمام رد و بدل اطلاعات CPU درگیر می شود و نمی‌تواند به دیگر پروسه‌ها جواب دهد ولی با DMA، پردازنده در شروع ارتباط درگیر خواهد شد سپس ارتباط Device و Memory به صورت مستقیم بر قرار می‌گردد سپس پس از پایان ارتباط یک Interrupt از چیپ DMA Controller دریافت می کند.

منابع

1. https://www.gta.ufrj.br/ensino/CPE758/artigos-basicos/IntelVT-Computer.pdf
2. https://blog.actorsfit.com/a?ID=01700-19a2ebb5-4cd5-4b0f-8888-17ff333e5184
3. https://www.gta.ufrj.br/ensino/CPE758/artigos-basicos/IntelVT-Computer.pdf
4. https://www.slideshare.net/xen_com_mgr/introduction-of-amd-virtual-interrupt-controller
5. https://www.semanticscholar.org/paper/vINT%3A-Hardware-Assisted-Virtual-Interrupt-Remapping-Li-Ma/d712373fea994375b27b983add3dae9333794c89
6. https://techterms.com/definition/irq
7. https://hardware.tosinso.com/fa/articles/40073/%D9%88%D9%82%D9%81%D9%87-Interrupt-%DA%86%DB%8C%D8%B3%D8%AA%D8%9F-%D9%85%D8%B9%D8%B1%D9%81%DB%8C-%D9%85%D9%81%D9%87%D9%88%D9%85-IRQ-%D8%AF%D8%B1-%D9%BE%D8%B1%D8%AF%D8%A7%D8%B2%D8%B4-%D8%B3%D8%AE%D8%AA-%D8%A7%D9%81%D8%B2%D8%A7%D8%B1