Are Virtual-Machine Monitors Microkernels Done Right?

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

Un artículo por Hand *et al.* en el reciente taller HotOS re-examina los microkernels y los contrasta con monitores de máquina virtual(*virtual machine monitors* - VMM). Se encuentra que los dos tipos de sistemas comparten criterios en arquitectura pero también tienen un número de diferencias técnicas las cuáles se examinan en el artículo. Se concluye que los VMMs son un tipo especial de microkernels, "microkernels bien hechos". Cuando se hace una mirada más cercana a los argumentos principales hechos por Hand *et al*, se encuentra que estos son difíciles de justificar o estan en desacuerdo con la literatura disponible.

Background. *History Revisited:* Ambos, Microkernels y VMMs tiene una larga historia que data de los años 70s. Golberg define un VMM como "*software que transforma la interfaz de una sola máquina en una ilusión de muchas. Cada una de etas interfaces (máquinas virtuales) es una replica eficiente del sistema original de la computadora completa con todas las instruciones de procesador ...*". Liedtke describe el enfoque microkernel como "*minimizar el kernel e implementar todo lo posible fuera del kernel*". Examinando los objetivos de los dos enfoques se muestra que hay más similitud que la evidente a partir de las definiciones: Goldber lista confiabilidad del software, seguridad de datos, APIs de sistema alternativas y nuevos mejoras y mecanismos como beneficios. Liedtke menciona flexibilidad, mantenibilidad e interdependencia restringuida. Parece qeu mientras los VMMs y microkerneles comparten un conjunto de objetivos, ellos toman un enfoque diferente hacia la solución. Ambos consideran la minimalización como algo importante. Mientras que para los microkernels este es un objetivo clave, Goldberg lo reporta como un resultado de la estructura del sistema: "*un principio clave en el análisis de la confiabilidad del software es que el VMM problemente esté correcto - la probabilidad de error es cercana a cero. Esta suposición es razonable porque el VMM probablemente sea un programa pequeño ...*".

Core primitives: en un esfuerzo por minimizar la funcionalidad del kernel, microkernesl ofrecen un conjunto mínimo de abstracciones con una primitiva central para extensibilidad: comunicación inter-proceso(IPC). En un microkernel, IPC provee tres propósitos primarios: (1) IPC es el mecanismo para cambio controlado por kernel de flujos de ejecución entre dominios de protección. (2) IPC es el mecanismo para transferencia de datos controlado por kernel entre dominios de protección. (3) IPC es el mecanismo para delegación de recursos entre dominios de protección que requiere acuerdo mutuo entre múltiples partes. Combinando estas tres operaciones ortogonales en una sola primitiva se reduce el número de mecanismos de seguridad, se reduce la complejidad del códiso y se reduce el tamaño del software. Un código más pequeño reduce el número de errores en el kernel privilegiado y también reduce la huella de caché. Un requerimiento obvio clave para cualquier microkernel es por tanto una primitiva de IPC de bajo *overhead*. Todas las otras tres operationces que requieren una combinación de los tres mecanismos puede ser implementados a través de una sola primitiva de IPC. VMMs, en comparación, se parecen mucho al hardware del procesador y ofrece una rica variedad de primitivas. Cada primitiva requiere de un conjunto dedicado de mecanismos de seguridad, recursos y código de kernel. A continuación una lista del

subconjunto común de primitivas que pueden encontrarse en la mayoría de VMMs: (1) cambio sincrónico de dominio de protección de usuario invitado(*user guest*) a kernel invitado(*guest kernel*). (2) cambio sincrónico de dominio de protección de *guest kernel* a *guest user*. (3) Canales de comunicación asincrónica a través de dominios (de máquina virtual(VM) a máquina virtual). (4) Asignación de recursos por VM a través de la interfaz*hypercall* de VMM. (5) Asignación de recursos dentro de la VM (via virtualización de *page-table* en hardware). (6) Re-asignación de recursos (via *page flipping*). (7) *Page-fault* y manejo de excepciones a través de virtualización de excepciones. (8) Notificación asincrónica de eventos a través de dominios via mecanismos de señales interrupcion-virtual. (9) Notificación de interrupción de hardware a través de un controlador de interrupciones virtualizadas. (10) Un conjunto de dispositivos comunes, tales como NIC y disco.

Las interfaces provistas por el VMM tienen un beneficio importante par una clase importante de software altamente complejo: los sistemas operativos existentes. Los SO disponibles ya programan contra la interface provista por el hardware y que ensambla el VMM. De esta forma los SO existentes pueda ser que no requieran cambios para correr en un VMM o bien, solamente algunos pocos. Esto requiere de significativa adaptación en las primitivas de microkernels. Sin embargo, este beneficio se está desgastando por el incremento en la divergencia de VMMs que van desde virtualización pura hasta paravirtualización.

La diversidad de interfaces también conduce a compromisos estructurales, tales como *super-VMs* centralizados que combinan y colocan funcionalidad crítica y significativa del sistema. Tal estructura puede disminuir potencialmente la confiabilidad y expopone el riesgo en un único punto de fallo. Este problema se convierte aun más inherente si esta *super-VM* corre en un SO legado el cual re-introducte un gran número de *bugs* de sofware.

Para extensiones que no están en un SO existente, la interface de VMM incrementa significativamente la complejidad del diseño del software. Dado que, por definición, un VMM presenta una interface que está cerca a la arquitectura subyacente, el software desarrollado para un VMM es inherentemente no portable entre arquitecturas. En contraste, un microkernel abstrae y oculta las pecularidades de la plataforma de hardware en un conjunto común de abstracciones. Por ejemplo, software que es escrito para un microkernel L4 naturalmente corre en nueve plataformas de procesador diferentes, desde dispositivo embebidos como ARM, sistemas de servidor o escritorio x86 hasta grandes multiprocesadores PowerPC e Itanium. Por lo tanto, es posible aprovechar y reusar los componentes de un sistema a traves de una amplia variedad de plataformas de hardware, minimizando el *overhead* en mantenimiento y en labores de conversión.

Architectural Lessons. *Avoid Liability Inversion:* Este artículo argumenta que mover servicios del sistema fuera del kernel relaja los límites de dependabilidad dentro del sistema. Aplicaciones e inclusive el kernel depende de código de nivel de usuario. Esta situación se llama inversión de responsabilidad y un se toma un ejemplo de Mach para argumentar que mecanismos inelegantes son requeridos para asegura el operación correcta del sistema como consecuencia de la renuncia del kernel a su responsabilidad. Luego se argumenta que una de las principales guías de diseño de Xen fue el evitar la inversión de responsabilidad.

En el taller, Butler Lampson señalo rápidamente que esta inversión de responsabilidad es de hecho un problema en Xen también. Un ejemplo de esto se da en otro artículo en el mismo taller por uno de los mismos autores: el sistema de almacenamiento Parallax usa páginas externas para proveer servicios de archivos. Mientras que el artículo artumenta que el diseño evita la inversión de responsabilidad, Parallax "provee un service de sistema crítico para un conjunto de VMMs". Esto es exactamente lo que un servidor de nivel de usuario hace en un sistema basado en microkernel. El argumento es que es un fracaso es que un fallo en el servidor Parallax solo afecta sus clientes - exactamente la misma situación que si un servidor falla un sistema basado en L4. Por lo tanto se falla en ver la diferencia entre un VMM y un microkernel en este particular. Posiblemente este aparente conflicto es un resultado de la falta de entendimiento de microkernels. La confusión puede ser de hecho el resultado de una generalización inválida de un ejemplo específico (una falla de diseño particular de Mach) con una clase enterra de sistemas (microkernels).

Make IPC performance irrelevant: En el artículo de Hand *et al.* se argumenta que mientras los diseñadores de microkernels han pasado un considerable esfuerzo en optimizar mecanismos de comunicación inter-procesos (IPC), esto es irrelevante dado que "no es una preocupación de diseño crítico en la construcción de VMMs de alto rendimiento". En el artículo se dice que IPC entre VMs es poco frecuente y por tanto no es crítico en rendimiento, como consecuencia de protección y *scheduling* de VMMs del SO. Esta es una línea argumental interesante y que está en desacuerdo con la realidad de sistemas basados en Xen en dos aspectos: (1) Xen usa VM separada (llamada Dom_0) para encapsular controladores de dispositivos legados. Por lo tanto, cualquier operación de entrada/salida implica al menos una comunicación ida y vuelta entre el VM *guest* y Dom_0. Los authores llamada a esto "mecanismo de eventos unidireccional asicrónico simple". Esto no es otra cosa que una forma de IPC asincrónico. (2) Mientras que es cierto que Xen planifica el SO completo, esto no es significa que no haya otra interacción con el VMM. De hecho, cada excepción de aplicaciones invitadas y llamados al sistema causa una *trap* dentro del VMM, lo que luego invoca funcionalidad correspondiente en el SO *guest*. Esto no es nada más que una operación de IPC entre la aplicación *guest* y el SO *guest*. Un sistema basado en Xen realiza esencialmente el mismo número de operaciones IPC que un sistema microkernel. *Treat the OS as a component:* Hand *et al.* argumenta que un beneficio de los VMMs es que están diseñados para correr un sistema legados completos, con programación familiar y ambientes de desarrollo. La - no declarada - implicación de tales declaraciones es que los microkernels son de alguna manera inadecuados para tal uso. Esta es una noción realmente sorprendente, dado que L4 ha demostrado muchos años atrás que es perfectamente adecuado como VMM soportando un sistema Linux paravirtualizado con excelente rendimiento, y el sistema Dresden DROPS se ha construido específicamente extendiendo a un sistema Linux paravirtualizado correiendo en un microkernel con servicios en tiempo real y que está en uso industrial. Nuevamente, se falla en ver la "diferencia significativa" entre VMMs y microkernels.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

El paper presentado por Hand *et al.* expone una serie de diferencias y beneficios de los VMMs por sobre los microkernels. Concluye diciendo que los VMMs son un caso especial de microkernels, "microkernels bien hechos". Una examinación más cercana de este artículo muestra que contiene un número de declaraciones que estan pobremente justificadas o inclusive refutadas por la literatura.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

En el artículo se identifican similitudes arquitecturales, se examinan diferencias entre los enfoques y se concluy que los VMMs son un tipo específico de diseño de microkernel, el "bien" hecho. No lo declara explícitamente pero hace una afirmación implícita que los VMMs tal como Xen son los únicos enfoques "adecuados" para construir microkernels.

Mirando de cerca las afirmaciones hechas por Hand *et al.*, se encuentra que estas son difíciles de justificar, o inclusive en desacuerdo con la literatura. Mientras que re-examinar los aciertos y desaciertos de los microkernels es una actividad valiosa, se recomienda que tales discusiones sean realizadas en concordancia con los principios científicos establecidos y estar respaldados por hechos. Como contribución a una discusión informada es que se examinan las afirmaciones hechas por Hand *et al.*

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

El trabajo previo en VMMs y microkernels data de los años 70s. Golberg y Liedtke proveen definiciones para VMMs y microkernels respectivamente. Aunque ambas tecnologías comparte un conjunto común de objetivos, ambos toman diferentes enfoques hacia la solución. En los dos enfoques la minimizalización es importante. Este es un tema clave en los microkernels pero según Goldberg en los VMMs este es el resultado de la estructura del sistema.

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

Diferencias señaladas por los autores con respecto al artículo de Hand *et al:*

- La diversidad de interfaces en VMMs puede llegar a colocar funcionalidad crítica significativa en un solo lugar (como en *super-VMs* centralizadas) lo cual llega a poner en riesgo ese único punto.
- Extensibilidad en las VMMs no es una tarea sencilla. Dependiendo de las interfaces que se implementen se podría llegar a construir software que no es portable.
- Aunque los microkernels dependan de código de nivel de usuario y se lleguen a consider una solución "poco elegante", la solución propuesta en el diseño de VMMs, la responsabilidad de inversión ha resultado ser un problema también, como Xen, un sistema basado en VMM.
- A pesar de que se considera el IPC como un aspecto poco relevante, en Xen se llevan a cabo muchas variedades de llamados IPC y realiza esencialmente el mismo número de operaciones IPC comparado con un sistema microkernel.
- L4, un sistema microkernel, es tan adecuado para soportar un sistema paravirtualizado y con entregar buen rendimiento como los VMMs.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

En resumen, las "diferencias importantes" entre los microkernels y VMMs identificados en Hand *et al.* no han sido sujeto de escrutinio. Como consecuencia, la conclusión de que los "VMMs son microkernels bien hechos" no puede ser deducido por los argumentos presentados. Por otro lado, la observación que los VMMs y los microkernels están muy relacionados merece más atención. También se cree que una examinación sistematizada y objetiva de las similitudes y diferencias de los microkernes y los VMMs está aun pendiente y podría ser una contribución muy valiosa en a la teoría y práctica de los SO.