Exokernel: An Operating System Architecture for Application-Level Resource Management

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

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

En la arquitectura **exokernel** las abstracciones tradicionales como memoria virtual y comunicación entre procesos, son implementadas enteramente a nivel de aplicación por *software* no confiable. In esta arquitectura, un kernel mínimo multiplexa los recursos de *hardware* disponible de forma segura. La librería de SO que trabaja por encima de la interfase de *exokernel* implementa abstracciones de mayor nivel. Los escritores de aplicaciones seleccionan librerías o implementan las suyas. Nuevas implementaciones de librerías de SO son incorporadas re-enlazando los ejecutables de aplicación. *Exokernel* provee una control a través de una interfase de bajo nivel. Un diseñador *exorkernal* tiene una meta: separar la protección de la administración. Para esto se "exportan" recursos de *hardware* en lugar de emularlos. Las tres técnicas para exportar recursos de forma segura son: *secure binding, visible resource revocation* y *abort protocol*.

The Cost of fixed high-level abstraction. Abstracciones en SO tradicionales tienden a ser muy generales, intentan proveer todas las funciones que se necesitan a todas las aplicaciones. Las abstracciones de alto nivel: (1) dañan el rendimiento porque no hay una forma de implementar una abstracción que sea la mejor para todas las aplicaciones. Obligan al SO a tener que hacer decisiones de direcciones de espacio y cargas de trabajo que pueden penalizar a las aplicaciones. (2) Escoden información a las aplicaciones, esto hace que sea difícil o imposible para las aplicaciones implementar sus propias abstracciones de gestión de recursos. (3) limitan la funcionalidad de aplicaciones porque ellas son las únicas interfases disponibles entre aplicaciones y recursos de *hardware*.

Exokernels: An End-to-End Argument. Las aplicaciones conocen mejor que los sistemas operativos cuáles son sus metas en decisiones de gestión de recursos y por lo tanto, se les debería dar tanto control como sea posible sobre estas decisiones. *Exokernel* permite que las abstracciones tradicionales sean implementadas enteramente a nivel de aplicación. Esta arquitectura consiste de una capa fina que multiplexa [1] y exporta recursos físicos de forma segura a través de un conjunto de primitivas de bajo nivel. Librerías de SO las cuales usan la interfase de bajo nivel de *exokernel*, implementan abstracciones de alto nivel y pueden definir implementaciones especiales que pueden cumplir mejor con las metas de rendimiento y funcionacilidad de las aplicaciones. **Library Operating Systems:** las aplicaciones que se ejecutan en *exokernel* puede reemplazar y extender libremente librerías de SO sin privilegios especiales, lo que simplifica el desarrollo y el agregar nuevas características. **Exokernel design:** al separar la protección de la administración se realizan 3 tareas: (1) rastreo del propietario de los recursos, (2) asegurar protección al custodiar todo el uso del recurso y (3) revocar acceso a los recursos. Las tareas se logran por 3 técnicas: *secure binding, visible revocation* y *abort protocol*. **Principios de diseño:** Exposición segura de

[1] Combinación de dos o más canales de comunicación en un sólo medio de transmisión.

hardware: sólo gestiona recursos en la medida requerida por la protección. Exponer la asignación: los recursos no son asignados de forma implícita, la librería de SO participa en la asignación. Exponer nombres: exportar nombres físicos (más eficientes). Exponer revocación: un protocolo de revocación de recursos visible para las librerías de SO. Policy: *exokernel* incluye un política para arbitrar entre librerías de SO competidoras: determinar la importancia y los recursos compartidos. **Secure bindings**: mecanismo de protección que desacopla la autorización del uso real del recurso. Son operaciones simples que el kernel ejecuta rápido y se llevan a cabo durante el tiempo de enlace (*bind time*). *Software* a nivel de aplicación puede usar primitivas de *hardware (TLB entry)* o *software* para soportar *secure bindings*. Se implementa por: *hardware*, *software caching* y descargando código de aplicación. **Visible Resource Revocation:** una vez que los recursos son enlazados tiene que haber una forma de reclamarlos y romper su *secure binding*. *Exokernel* usa revocación visible – le informa a las aplicaciones y estas pueden reaccionar. **The Abort Protocol:** si una librería de SO falla en responder rápido los enlances seguros (*secure bindings*) necesitan ser rotos "a la fuerza". Estas pérdidas forzadas de un recurso se registran en un vector de reposición que se puede usar para actualizar los mapeos hacia ese recurso.

Aegis: an Exokernel. Aegis: implementación de un software que sigue la arquitectura de exokernel. Exporta el procesador, memoria física, TLB, excepciones e interrupciones.

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

La interfase que definen los SO entre aplicaciones y recursos puede limitar significativamente el rendimiento y la libertad de implementación de las aplicaciones. Tradicionalmente, los SO esconden información acerca de los recursos de la máquina detrás de abstracciones de alto nivel como procesos, archivos, espacios de direcciones y procesos de intercomunicación. Estas abstracciones definen una máquina virtual en donde las aplicaciones se ejecutan; su implementación no puede ser reemplazada o modificada por aplicaciones no confiables. "Hardcodear" la implementación de estas abstracciones es inapropiado por: le niega a las aplicaciones las ventajas de las optimizaciones de dominio específico, se desalienta a cambiar las implementacione de abstracciones actuales y reestringuen la flexibilidad de constructores de aplicaciones dado que nuevas abstracciones son difíles de agregar.

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

Porque al delegar a las aplicaciones la implementación de estas abstracciones se puede tener un mejor control sobre la ejecución y el rendimiento de estas abstracciones.

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

Otros artículos y proyectos han discutido la necesidad de kernels extensibles y flexibles:

- Hydra fue el más ambicioso sistema en tener una separación entre la política y mecanismo del kernel como uno de sus principios centrales.
- VM/370: exporta la interfase de *hardware*. Por encima de esta interfase se soportan un número de máquinas virtuales en donde sistemas operativos radicalmente diferentes pueden ser implementados. Esta flexibilidad se provee a partir de la virtualización de toda la máquina base.
- Microkernels: similitud con exokernels en que están diseñados para incrementar la flexibilidad. Diferencia en que los exorkernels ponen la interfase del kernel mucho más cerca al hardware, lo cual brinda mayor flexibilidad.
- SPIN: un microkernel qeu permite a las aplicaciones hacer decisiones sobre políticas por medio de descargas seguras de extensiones al kernel. A diferencia de SPIN, la meta de exokernel es obtener flexibilidad y rendimiento por medio de la exposición de primitivas de bajo nivel en lugar de extender SO tradicionales en una forma segura.

- SPACE: un "submicro-kernel" que provee solamente abstracciones de bajo nivel definidas por la interfase de arquitectura y de trampa.

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

La arquitectura *exokernel*. Un *exokernel* multiplexa los recursos de *hardware* disponibles entre las aplicaciones de forma segura. Las librerías de SO, las cuales trabaajan por encima de la interfase de bajo nivel de *exokernel*, implementan abstracciones de alto nivel y pueden definir implementaciones para propósitos especiales las cuales cumplan con las metas de rendimiento y funcionalidad de las aplicaciones de mejor forma. Esta arquitectura está motivada por una simple observación: entre más bajo sea el nivel de la primitiva, mayor será la eficiencia en su implementación. Para lograr esta interfase de bajo nivel, *exokernel* separa la gestión de la implementación. Para hacer esta separación eficiene se usa *secure bindings*, implementados usando mecanismos de *hardware*, *software caches* o descargando código.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

Se implementa un prototipo de un sistema operativo tipo *exokernel* basado en *secure bindings*, revocación visible y protocolos de anulación (*abort*). Aegis es la implementación del *exokernel* y ExOS un librería de SO no confiable (*untrusted*). Los experimientos con estos prototipos demostraron las siguiente hipótesis:

1. la simplicidad y el número limitado de primitivas *exokernel* les permite que sean implementadas muy eficientemente. Las mediciones en Aegis mostraron que sus primitivas básicas son substancialmente más eficientes que las primitivas generales provistas por en Ultrix. Además, el rendimiento de Aegis es mejor o similar con implementaciones recientes de alto rendimiento de despacho de excepciones y control de transferencia de primitivas.
2. Debido a que las primitivas de *exokernel* son rápidas, la multiplexación segura de bajo nivel de recursos de hardware puede ser implementada de forma eficiente. Por ejemplo, Aegis multiplexa recursos como el procesador, memoria y de red más eficientemente que las implementaciones más recientes.
3. Abstracciones tradicionales de SO pueden ser implementadas eficientemente a nivel de aplicación. Por ejemplo, la memoria virtual de nivel de aplicación y las primitivas de comunicación interprocesos en ExOS son mucho más rápidas que las de Ultrix y de las implementaciones más recientes.
4. Las aplicaciones pueden crear implementaciones especiales de las abstracciones simplemente modificando una librería. Se implementaron variantes de abstracciones fundamentales de SO como la de comunicación interproceso, memoria virtual y *schedulers* obteniendo mejoras substanciales en funcionalidad y rendimiento.