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

Los sistemas operativos(SO) definen una interfase entre aplicaciones y recursos físicos. Desafortunadamente, esta interfase 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.

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 *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.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?
2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?
3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?
4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?
5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

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