Exokernel

An Operating System Architecture for Application-Level Resource Managment

E. Mancuso
A. Mataloni
M. Miguel

16 de Junio de 2011

E. Mancuso
A. Mataloni
M. Miguel

1 Introducción
Kernel Monolítico
Exokernel
Arquitectura
Ventajas
2 Diseño
Técnicas
Secure bindings
Visible resource revocation
Abort protocol
3 Experimentos
Excepciones
IPC
Memoria virtual
Flexibilidad
4 Conclusión

Introducción

La mayoría de los SO actuales utilizan kernels monolíticos, esto implica

- Interfases de hardware genéricas
- Poca flexibilidad para la gran cantidad de aplicaciones existentes
- Implementaciones Ad-Hoc de abstracciones (Procesos, IPC, Archivos, etc)
- Las aplicaciones corren en máquinas virtuales cuya base son abstracciones de alto nivel
- Ocultar información del Hardware a la aplicación
- Frena la innovación en la implementación de abstracciones

Es muy difícil implementar una abstracción que de un excelente rendimiento para toda aplicación existente y por ser desarrollada.

Por eso el exokernel propone cambiar la arquitectura monolítica por la siguiente

- **Exokernel**: multiplexa los servicios de hardware de manera segura mediante primitivas de muy bajo nivel
- **Biblioteca SO**: Implementacion de abstracciones de alto nivel en espacio de usuario.

E. Mancuso
A. Mataloni
M. Miguel

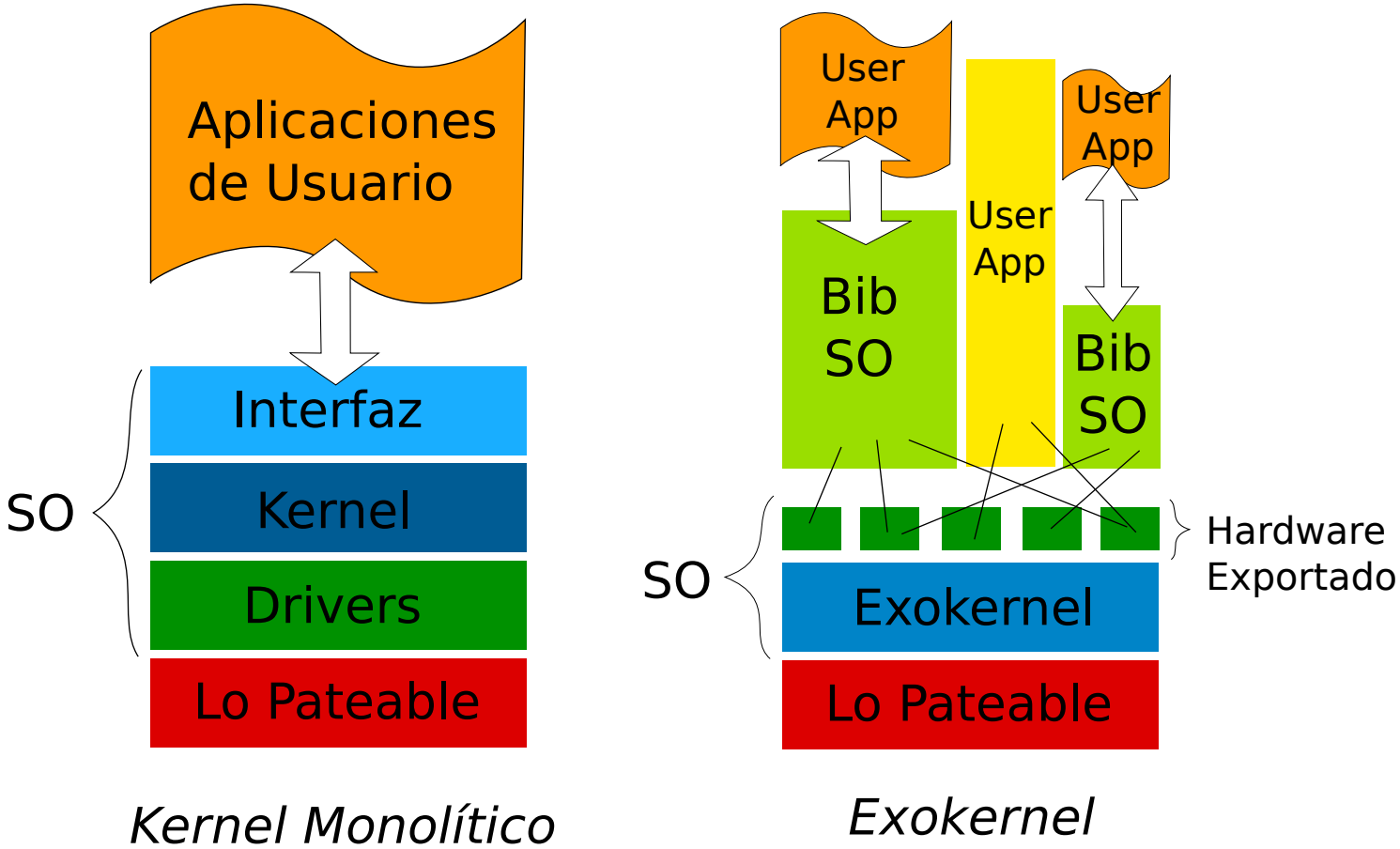

Figura: Gráfico comparativo de los módulos en un kernel monolítico y un exokernel

- *A menor nivel de la primitiva, más eficientemente puede esta implementarse*
- Interfaces más específicas
- Exportar el Hardware al usuario (programador de aplicaciones) en lugar de emularlo
- Se hace visible todo estado del hardware y se exportan las instrucciones privilegiadas (con protecciones)
- La interfaz se define en el espacio de usuario
- Controlar recursos desde nivel de usuario
- Las aplicaciones conocen mejor que el SO qué recursos necesitan y cómo administrarlos

E. Mancuso
A. Mataloni
M. Miguel

Diseño

Un **exokernel** se encarga de tres importantes tareas

- Seguimiento de la asignación de recursos
- Garantizar la protección del uso de recursos o *binding points*.
- Revocar el acceso a los recursos

Para lograr implementar esta arquitectura hay que conocer las tres técnicas que utiliza el **exokernel**

- Secure bindings
- Visible resource revocation
- Abort protocol

Secure bindings

Es un mecanismo de protección que desacopla autorización del uso real de un recurso.

Las bibliotecas de Sistema Operativo pueden unirse o *'bindiarse'* al Hardware para tener un mayor control sobre los eventos.

La autorización se realiza sólo en el *binding time*, esto permite separar la administración de la protección.

El **exokernel** interviene en todos los accesos a recurso.

Visible resource revocation

Como parte del concepto de *que el usuario sabe manejar sus recursos mejor*, se permite a las **bibliotecas de sistema operativo** participar en el protocolo de revocación de recursos.

Esto significa que, cada vez que el **exokernel** necesita que se liberen recursos, el protocolo para lograr la liberación implica una conversación entre el kernel y la **biblioteca SO**. En el mismo, se hace un pedido de liberación desde el kernel al nivel de aplicación.

Abort protocol

Es un protocolo para romper, a la fuerza, *secure bindings* con las **bibliotecas de sistema operativo** que no cooperan o fallan al responder una solicitud de revocación.

El **exokernel** rompe todos los *secure bindings* existentes con el recurso e informa a la **biblioteca de sistema operativo**. Ésta acción es registrada en un *vector de recuperación* y la biblioteca recibe una excepción de **recuperación** para actualizar a quienes utilizan el recurso.
A las **bibliotecas de sistema operativo** se les asegura un mínimo de recursos que no le serán revocados.

Experimentos

Para ver que esto efectivamente puede suceder, se implemento **Aegis** un exokernel y **ExOS**, una biblioteca de sistema operativo que implementa las abstracciones más importantes de los sistemas operativos.
Los experimentos prueban estas hipótesis

- Los **Exokernels** pueden ser muy eficientes
- La multiplexación segura de hardware se puede implementar de manera eficiente a bajo nivel.
- Las abstracciones de SO tradicionales se pueden implementar eficientemente a nivel de aplicación.
- Las aplicaciones pueden crear implementaciones de estas abstracciones con propósito específico.

Excepciones

Machine	OS	unalign	overflow	coproc	prot
DEC2100	Ultrix	n/a	208.0	n/a	238.0
DEC2100	Aegis	2.8	2.8	2.8	3.0
DEC3100	Ultrix	n/a	151.0	n/a	177.0
DEC3100	Aegis	2.1	2.1	2.1	2.3
DEC5000	Ultrix	n/a	130.0	n/a	154.0
DEC5000	Aegis	1.5	1.5	1.5	1.5

Cuadro: Tiempo de envío de excepciones en **Aegis** y **Ultrix** (tiempos en milisegundos).

En **Aegis**, al producirse una excepción, el estado se guarda en un área de memoria ya acordada con el usuario, con accesos directos a memoria física. Esto evita *TLB misses*.
Luego se ejecuta el controlador de interrupciones de la aplicación, dejando a su disposición la información necesaria para que pueda reconstruir el estado necesario y vuelva a ejecutar sin pasar nuevamente por el kernel.

IPC

Machine	OS	pipe	pipe'	shm	lrpc
DEC2100	Ultrix	326.0	n/a	187.0	n/a
DEC2100	ExOS	30.9	24.8	12.4	13.9
DEC3100	Ultrix	243.0	n/a	139.0	n/a
DEC3100	ExOS	22.6	18.6	9.3	10.4
DEC5000	Ultrix	199.0	n/a	118.0	n/a
DEC5000	ExOS	14.2	10.7	5.7	6.3

Cuadro: Comparativas de implementaciones de **IPC** en **ExOS** sobre **Aegis** y **Ultrix** (tiempos en milisegundos). Para pipe y memoria compartida las mediciones son unidireccionales, para LRPC es bidireccional.

En estos experimentos se hace una prueba de ping-pong entre dos procesos actualizando un contador. **pipe'** es una implementación más eficiente de **ExOS** de pipes. En **lrpc** se mide el tiempo de hacer una llamada remota, actualizar un contador en otro espacio de direcciones y volver al contexto original.

Memoria virutal

Machine	OS	dirty	prot1	prot100	unprot100	trap	appel1	appel2
DEC2100	Ultrix	n/a	51.6	175.0	175.0	240.0	383.0	335.0
DEC2100	ExOS	17.5	32.5	213.0	275.0	13.9	74.4	45.9
DEC3100	Ultrix	n/a	39.0	133.0	133.0	185.0	302.0	267.0
DEC3100	ExOS	13.1	24.4	156.0	206.0	10.1	55.0	34.0
DEC5000	Ultrix	n/a	32.0	102.0	102.0	161.0	262.0	232.0
DEC5000	ExOS	9.8	16.9	109.0	143.0	4.8	34.0	22.0

Cuadro: Comparativa de tiempos de operaciones de memoria virtual en **ExOS** y **Ultrix** (Tiempos en milisegundos). **appel1** y **appel2** son benchmarks de *Appel and Li* y el tiempo presentado promediado por página.

Flexibilidad

Se presentan los beneficios de trabajar con distintas implementaciones de la mismas abstracciones.

Machine	Method	dirty	prot1	prot100	unprot100	trap	appel1	appel2
DEC2100	Original page-table	17.5	32.5	213.	275.	13.9	74.4	45.9
DEC2100	Inverted page-table	8.0	23.1	253.	325.	13.9	54.4	38.8
DEC3100	Original page-table	13.1	24.4	156.	206.	10.1	55.0	34.0
DEC3100	Inverted page-table	5.9	17.7	189.	243.	10.1	40.4	28.9

Cuadro: Operaciones de memoria virtual usando distintas estructuras de tablas de página (tiempos en milisegundos).

Machine	lrpc	tlrpc
DEC2100	13.9	8.6
DEC3100	10.4	6.4
DEC5000	6.3	2.9

Cuadro: Comparación de ejecuciones de *lightweight remote procedure call* contra *trusted lightweight remote procedure call* (tiempos en milisegundos).

Conclusión

- El **exokernel** permite crear abstracciones de alto nivel especializadas para cada tipo de aplicación. Esto lleva a una mejora en rendimiento y flexibilidad.
- Los experimentos realizados sobre las implementaciones **ExOS** y **Aegis** demostraron las hipotesis.
- Basado en los resultados, se concluye que la arquitectura del **exokernel** es una estructura viable para la implementación de sistemas operativos extensibles y con muy buen rendimiento.