Extensibility, Safety and Performance in the *SPIN* Operating System

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

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

SPIN es un sistema operativo que puede brindar una especialización dinámica para cumplir con requerimientos de rendimiento y funcionalidad de aplicaciones de forma segura. Está motivado por la necesidad de soportar aplicaciones que presentan demandas que no son igualadas por la implementación de un sistema operativo (SO) o interfase. Una implementación mal adaptada evita que una aplicación trabaje bien, mientras que una interfase mal adaptada evita que una aplicación trabaje por completo. Por ejemplo, una implementación de almacenamiento y algoritmos de paginación en disco que se encuentra en sistemas operativos modernos pueden ser inapropiados en aplicaciones de bases de datos, dando como resultado un rendimiento pobre.

Con *SPIN*, una aplicación puede extender las interfases e implementaciones del SO para proveer una solución que se adapta mejor a las necesidades de la aplicación, rendimiento y características funcionales del sistema.

Goals and approach. Extensibilidad: determinada por las interfases a los servicios y recursos que son exportados por la aplicaciones. Seguridad: determina el nivel exposición de las aplicaciones a las acciones de otros, y requiere que el acceso sea controlado con la mismo nivel de granularidad con el que las extensiones fueron definidas. Rendimiento: requiere una comunicación con un bajo *overhead* entre una extensión y el sistema. *SPIN* depende de 4 técnicas implementadas a nivel de lenguaje on su ejecución: (1) *Co-location:* las extensiones de SO son enlazadas dinámicamente dentro del espacio de direcciones virtuales del kernel. *Co-location* permite que la comunicación entre el sistema y el código de la extensión tenga un bajo costo. (2) *Enforced modularity:* las extensiones están escritas en Modula-3. Se ejecutan en el espacio de memoria virtual del kernel. No puede ejecutar accesos a memoria o instrucciones privilegiadas a menos de que se les haya dado acceso explícico a través de la interfase. La modularidad forzada por el compilador permite que las extensiones estén aisladas unas de otras. (3) *Logical protection domains:* las extensiones existen dentro de dominios lógicos protegidos, los cuales son espacios de nombres del kernel que contienen código y exportan interfases. Las interfases representan vistas de recursos del sistema que están protegidos por el SO. (4) *Dynamic Call binding:* las extensiones se ejecutan en respuesta a eventos del sistema. Un evento puede describir cualquier acción potencial en el sistema. Los eventos se declarados en interfases y pueden ser despachadas con el mismo *overhead* de una llamada a un procedimiento.

System Overview. *SPIN* consiste de un conjunto de servicios de extensión y de servicios básicos(*core*) del sistema que se ejecutan dentro del espacio de direcciones virtuales del kernel. Las extensiones pueden ser cargadas en el kernel en cualquier momento. Una vez cargadas, se integran con la infraestructura existente y proveen servicios específicos del sistema a las aplicaciones que lo requiran. *SPIN* está escrito principalmente en Modula-3, lo que le permite a las extensiones el uso directo de interfases del sistema sin requerir conversiones en tiempo de ejecución cuando se comunican con otro código del sistema. Aunque *SPIN* depende de las capacidades del

lenguaje para asegurar seguridad dentro del kernel, las aplicaciones pueden ser escritas en cualquier otro lenguaje y ejecutarse dentro de su propio espacio virtual de direcciones. Solamente el código que requiera acceso de baja latencia al los servicios del sistema son escritos en el lenguaje de extensión seguro del sistema.

The *SPIN* Architecture. La arquitectura provee una infraestructura de *software* para combinar código de sistema y de aplicación de forma segura. El modelo de protección brinda un control al acceso de los recuros de forma segura y detallada. El modelo de extensión permite que las extensiones sean definidas con la granularidad de una llamada a un procedimiento. Sobre Modula-3: Una interfase declara las partes visibles de la implementación de un modulo. Tipos seguros *type safety* previenen que el código tenga acceso a la memoria de forma arbitraria. Gestión de almacenamiento automático evita que la memoria usada por un puntero sea retornada al *head* y reusada por un objeto de tipo diferente. **The protection model**: controla el conjunto de operaciones que pueden ser aplicadas a los recursos. Por ejemplo, un modelo de protección basado en espacioes de memoria asegura que un proceso solo puede tneer acceso a memoria dentro un rango particular de direcciones virtuales. Capabilities: Todos los recursos del kernel estan referenciados por *capabilities*. Un *capability* es una referencia inolvidable hacia un recurso que puede ser un objeto del sistema, interfase o colección de interfases. *SPIN* implementa *capabilities* por medio del uso directo de punteros, los cuales son soportados por el lenguaje. El compilador, en tiempo de compilación, evita que un puntero sea modificado o de-referenciado de forma inconsistente a su tipo. Un puntero puede ser pasado a una aplicación de usuario desde el kernel, una referencia externa la cual no puede ser asumido que sea un tipo seguro. Esta referencia es un índice hacia una tabla que contiene referencias a tipos seguros en estructuras de datos en el kernel. Las referencias pueden ser recuperadas luego por medio del índice. **Protection domains:** define un conjunto de nombres disponibles accesibles en un contexto de ejecución. En un SO convencional, el dominio de protección se implementa usando espacioes de direcciones virtuales. Un nombre dentro un dominio, no tiene relación con el mismo nombre en otro dominio. Solamente a través de mapeo explícito y operaciones compartidas es posible hacer que los hombres sean significativos entre dominios de protección. El dominio de protección de *SPIN* define un conjunto de nombres que pueden ser referenciados por código con acceso al el dominio. Un dominio, es usado para controlar el enlace dinámico y corresponde a uno o más archivos de objeto seguro con una o más interfase exportada. Un archivo de objeto seguro puede ser desconocido para el kerkel pero ha sido firmado por el compilador de Modula-3, lo cual lo convierte en seguro. *Create*: operación par crear un dominio, inicializa un dominio con los contenidos de un archivos de objecto seguro. Se exportan símbolos definidos por la interfase al dominio. *Resolve:* operación que sirve como la base para el enlazado dinámino. Resuelve cualquier símbolo no resuelto en el dominio objetivo (*target*) contra símbolos exportados desde la fuente. *Combine*: crea espacios de nombres enlazables que son la unión de dominios existentes, y pueden ser usados para enlazar colecciones de interfases relacionadas. **The extension model:** provee la habilidad de una comunicación controlada entre las extensiones y el sistema base, mientras permite un variedad de estilos de interacción. Las extensiones son definidas en términos de eventos y manejadores. Un evento es un mensaje que anuncia un cambio en el estado del sistema. Un manejador es un procedimiento que recibe el mensaje. Una extensión registra explícitamente un manejador con un evento a través de un despachador central el cual dirige eventos hacia manejadores. Un evento es definido como un procedimiento exportado desde una interfase y su manejador es definido como procedimientos que tienen el mismo tipo. Un manejador es invocado con los argumentos especificados por el generador del evento.

The core services. *SPIN* provee un conjunto de servicios básicos que gestionan recursos de memoria y procesador. Estos servicios usan eventos para comunicarse entre el sistema y extensiones, exportan interfases con operaciones detalladas. **Extensible memory management:** un sistema de gestión de memoria es responsable por la asignación de direcciones virtuales, físicas y los mapeos entre las dos. Componentes de la interfase de memoria de *SPIN: Physical address service:* controla la asignación y uso de páginas físicas. *Virtual address service:* asigna *capabilities* únicas para direcciones virtuales. *Translation service:* usado para expresar la relación entre

direcciones virtuales y memoria física. **Extensible thread management**: el sistema de gestión de hilos de un SO provee a las aplicaciones de interfases para *scheduling*, concurrencia y sincronización. Las aplicaciones pueden requerir niveles de funcionalidad y rendimiento que el sistema gestor de hilos no puede entregar. En *SPIN* una aplicación puede proveer su propio paquete de hilos y *scheduler* que son ejecutados dentro del kernel. Aunque no se define un modelo de hilos para las aplicaciones, se define una estructura en donde una implementación de un modelo de hilos reside. Esta estructura es definida por una serie de eventos que son generados o manejados por *schedulers* y paquetes de hilos. Un *scheduler* multiplexa los recursos de procesamiento subyacente con contextos competidores llamados *strands*. A diferente de un hilo, un *strand* el único requisito mínimo es un nombre. El paquete específico de hilos de una aplicación define una implementación de *strands* para us propios hilos. **Implications for trusted services:** Los servicios de procesador y memoria son dos instancias de servicios básicos de *SPIN*, lo cuales proveen interfases para mecanismos de hardware. Los servicios básicos son confiables lo que significa que ellos van a comportarse de acuerdo a la especificación de su interfaz. Confianza es requerida porque los servicios requieren acceso a hardware y a veces deben de operar fuera del modelo de protección forzado por e lenguaje. Son confianza, los mecanismos de protección y extensión no podría funcionar de forma segura ya que se depende de una apropiada gestión del *hardware*. Debido a que los servicios confianbles actúan como mediadores para el acceso de recursos físicos, las aplicaciones y las extensiones deben confiar en los servicios que son de confianza en el kernel de *SPIN*.

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

La necesidad de brindar a las aplicaciones soporte para que puedan llevar a cabo demandas que no son igualadas por la implementación de un sistema operativo (SO) o interfase.

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

Una implementación mal adaptada evita que una aplicación trabaje bien, mientras que una interfase mal adaptada evita que una aplicación trabaje por completo. Por ejemplo, una implementación de almacenamiento y algoritmos de paginación en disco que se encuentra en sistemas operativos modernos pueden ser inapropiados en aplicaciones de bases de datos, dando como resultado un rendimiento pobre. Implementaciones genéricas de red son frecuentemente inadecuadas para llevar a cabo tareas en aplicaciones paralelas de alto rendimiento. También en aplicaciones multimedia, servidores, tiempo real y tolerancia a fallos son escenarios en donde los SO no proveen servicios adecuados.

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

- Hydra: define una infraestructura que permite a las aplicaciones gestionar recursos a través de políticas multi-nivel. El kernel define mecanismos para asignar recursos entre procesos y los procesos implementan las políticas para administrar esos recursos. La arquitectura de Hydra ha sido muy influyente pero tiene un alto *overhead* debido a su pesado mecanismo de protección basado en *capabilities*. En consecuencia el sistema ha sido diseñado con “objetos grandes” como sus bloques básicos de contrucción requiriendo un gran esfuerzo en programación para afectar inclusivo a una pequeña extensión.
- Microkernels: un microkernel exporta típicamente un pequeño número de abstracciones. Estas abstracciones puede ser combinadas para ayudar a que servicios en SO convencionales puedan ser implementados como programas de nivel de usuario. Desafortunamente las aplicaciones requiren cambios substanciales dentro de una implementación de microkernel para compensar la limitación de interfases.
- Algunos sistemas dependen de “lenguajes pequeños” para extender la interfase del SO de forma segura. Tiene los siguientes problemas: la expresividad de estructura arbitrarias de control y datos se vuelve incómoda, esto limita el rango de extensiones posibles. La interfase entre el ambiente del SO y el lenguaje

de programación es generalmente angosta, lo que hace la integración dificultosa. El *overhead* en la implementación puede limitar el rendimiento.
- Aegis: un SO que depende de una redirección de trampa eficiente para exportar servicios de *hardware* directamente en las aplicaciones. El sistema como tal no define abstracciones más alla de las mínimas provistas por el *hardware*. En su lugar, sistemas convenciales de SO son implementados como librerías que se ejecutan en el espacio de direcciones de la aplicación. El código de servicios de sistema ejecutando una librería puede ser cambiado por la aplicación de acuerdo a sus necesidades.

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

La arquitectura *SPIN*, un sistema operativo extensible. Provee una infraestructura extensible junto con una serie de servicios básicos extensibles, que permiten a las aplicaciones cambiar la interfase e implementación del SO. Las extensiones permiten a la aplicación a especializar el SO subyacente con el fin de lograr un nivel particular de rendimiento y funcionalidad.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

Se evaluó *SPIN* desde las siguientes perspetivas:

- Tamaño del sistema: en términos de líneas de código y el tamaño de los objetos se demostró que los servicios no necesariamente crean un kernel de SO de tamaño excesivo. Además, el tamaño del sistema de extensiones muestra que ellos pueden ser implementadas con una cantidad razonable de código.
- *Microbenchmarks*: las mediciones de los servicios de bajo nivel como comunicación protegida, gestión de hilos y memoria virtual, mostraron que la arquitectura extensible de *SPIN* permite construir servicios de comunicación intensiva con bajo *overhead*. Las mediciones tambien muestran que los mecanismos de sistemas convencionales como llamadas al sistema y llamados a procedimientos de espacio protegidos tienen un *overhead* comparable con los mismos en SO convencionales.
- *Networking:* Las mediciones de un conjunto de protocolos de red demostraron que la arquitectura extensible de *SPIN* permite la implementación de protocolos de red de alto rendimiento.
- *Ent-to-End performance:* se mostró a partir de la implementación de dos aplicaciones(un sistema cliente/servidor de video y un *Web Server*) que el rendimiento de completo de una aplicación se puede beneficiar de *SPIN*.