Diseño de un sistema operativo reconfigurable para fines didácticos y prácticos

S. M. Martin, G. E. De Luca, and N. B. Casas

***Abstract*— This paper contemplates the design view of an operating system that allows for managing multiple low-level configurations without the need of recompiling. Most of the common commercial Unix-based and Windows OSs are designed to run with a particular configuration for performance, storage optimization, and/or hardware compatibility. However, in order to prioritize the didactic value of an OS, it can be useful to let the students experiment with different architectural configurations and compare their behavior. Finally, we will use this perspective to describe a design path towards an adaptive architecture OS.**

***Keywords*— Didactic Approach, Operating Systems, Configurable Architecture, Adaptive Devices**

I. Introducción

El diseño de un sistema operativo depende en gran medida de las necesidades específicas del mercado al que se destinará su distribución, del entorno (hardware, peopleware, orgware, y otros componentes software) en el que funcionará, y de los estándares que se deban respetar.

Aspectos como la inicialización de dispositivos, administración de memoria, la planificación de procesos e hilos, la administración de la seguridad de permisos, las interfaces kernel/usuario, las interfaces con los dispositivos, y los sistemas de archivos soportados, pueden contar cada uno con decenas de posibles implementaciones (o modos) diferentes; encontrar la combinación correcta para las necesidades específicas a satisfacer es, por lo tanto, una tarea que requiere un análisis exhaustivo. Una mala decisión en al menos uno de estos aspectos puede causar el fracaso en el producto final [1] [2].

La elección de una de estas combinaciones, es decir, una implementación particular para cada uno de estos aspectos, da por resultado la configuración a bajo nivel del sistema operativo a desarrollar —nos referiremos a esto simplemente como configuración del sistema operativo—.

Desde del punto de vista del proceso de software, esta decisión se toma en la etapa de diseño, y marca el camino a seguir a los programadores para el desarrollo del sistema operativo.

Salvo contadas excepciones [3], no existe necesidad de desarrollar varias alternativas para implementar uno o varios de estos aspectos, ya que esto demandará recursos y tiempo de desarrollo adicional, no solo para cada alternativa, sino también para el mecanismo que permita alternar entre ellas sin necesidad de re-compilar.

Si bien pueden existir configuraciones completamente diferentes entre sistemas operativos para computadores personales, para servidores, de tiempo real, o tolerantes a fallos, ninguno de éstos requiere ser reconfigurable durante su ejecución. Existen, sin embargo, casos en los que se proveen herramientas para mantener una retrocompatibilidad hacia versiones anteriores con diferentes configuraciones [4].

Más allá del punto de vista productivo, el diseño de un sistema operativo que soporte la administración de múltiples configuraciones sin necesidad de recompilar puede resultar un buen recurso para docentes y alumnos para la comprensión de su funcionamiento a bajo nivel.

Un alumno puede experimentar el transcurso de interacciones, acciones en lote, o la ejecución de un proceso, observando diferentes resultados según la configuración que se haya elegido. Los diferentes resultados pueden incluir: falta/sobra de recursos, diferencias de rendimiento, errores y excepciones predecibles, abrazos mortales, entre otros. Esto ayuda al alumno a adquirir una visión general del diseño de sistemas operativos, teniendo en cuenta todas las alternativas en los aspectos que determinan su configuración.

A su vez, el docente tiene la posibilidad de enriquecer sus clases teóricas utilizando casos de prueba preparados para cada las diferentes implementaciones. Por ejemplo, se pueden analizar diferentes planificadores corriendo el mismo lote de procesos de prueba para cada uno con solo cambiar la variable de entorno correspondiente.

Desde el punto de vista didáctico, el factor distintivo del enfoque de diseño planteado en este artículo respecto a utilizar diferentes sistemas operativos de única configuración precompilados es el análisis *ceteris paribus* de los cambios efectuados sobre la configuración; todo, salvo lo que se cambia, se mantiene inalterado, aislando sus consecuencias y, por lo tanto, dando una idea más clara —o menos contaminada— de su funcionamiento.

En este artículo seguiremos la siguiente disposición: en la sección II daremos una introducción al proyecto SODIUM; en la sección III proporcionaremos los definiciones y conceptos que utilizaremos en el resto del trabajo; en la sección IV mostraremos el diseño reconfigurable de administración de memoria aplicadas en SODIUM a modo de ejemplo; en la sección V se analizará el diseño de un sistema operativo autoconfigurable y adaptable; y finalmente, en la sección VI, se encontrarán las conclusiones y un mapa de trabajo para la inclusión de nuevos aspectos hacia una arquitectura reconfigurable unificada en el futuro.

S. M. Martin, Universidad Nacional de La Matanza (UNLaM), Buenos Aires, Argentina, smartin@unlam.edu.ar

G. E. De Luca, Universidad Nacional de La Matanza (UNLaM), Buenos Aires, Argentina, gdeluca@unlam.edu.ar

N. B. Casas, Universidad Nacional de La Matanza (UNLaM), Buenos Aires, Argentina, ncasas@unlam.edu.ar

II. PROYECTO SODIUM

SODIUM (Sistema Operativo del Departamento de Ingeniería de la Universidad de la Matanza) es un proyecto de desarrollo conjunto entre alumnos y profesores de la materia Sistemas Operativos de la carrera de Ingeniería en Informática de la Universidad Nacional de La Matanza de un sistema operativo didáctico.

Se trata de un sistema operativo de multiprogramación basado en la arquitectura Intel IA32. Sumado a esto, las perspectivas a futuro incluyen soportar multiprocesamiento, e instrucciones de 64 bits.

Iniciado en el año 2005, y con el objetivo de darles una oportunidad a los alumnos, no solo de aprender los conceptos teóricos, sino de poder involucrarse de manera práctica del desarrollo de un sistema operativo, SODIUM fue evolucionando a partir de la entrega de trabajos prácticos consistentes en módulos agregables, por ejemplo, un driver de interfaz IDE o el mecanismo de arranque utilizando un pendrive.

A final de cada año, todos los trabajos prácticos son sometidos a diferentes casos de prueba, y los mejores son adaptados e integrados al sistema operativo creando una nueva versión base para el año siguiente.

Uno de los primeros dilemas surgidos de esta metodología de trabajo surgió, predeciblemente, cuando se solicitaron diferentes implementaciones de un aspecto particular del sistema operativo. Por ejemplo, contando con un SODIUM base con un administrador de memoria basado en particiones equitativas, un trabajo práctico destinado al desarrollo de un administrador basado en particiones variables o paginación, debía forzosamente reemplazar al existente. Llevar esto a cabo implicaba una pérdida de valor —didáctico— importante ya que, si bien la administración de particiones variables es superior productivamente [5], se estaría perdiendo la oportunidad de compararlas dentro de la misma compilación de SODIUM. Tampoco serviría utilizar las versiones base de otros años para efectuar la comparación, ya que se perdería el valor *ceteris paribus*, es decir, podrían haber cambiado también otros aspectos del sistema operativo que también hayan cambiado alterando el resultado de la prueba [ver Figura 1].

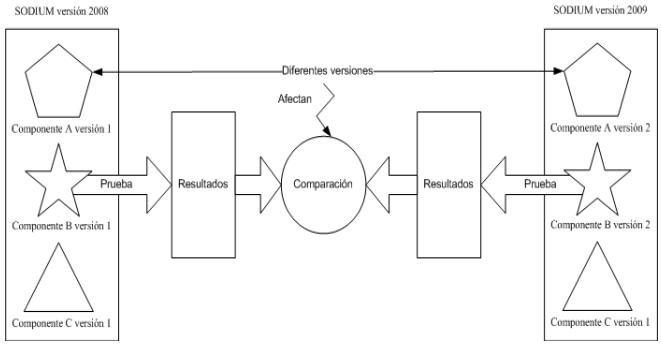

Figura 1. La comparación entre diferentes versiones del componente B es alterada por los cambios en A

Ante esta situación, los docentes de la cátedra tomaron la decisión de adoptar una arquitectura que tienda a maximizar la cantidad de aspectos del sistema operativo que puedan configurarse en tiempo de ejecución —los diferentes momentos en los que puede cambiarse un aspecto se describen en el capítulo III—.

Ambos trabajos prácticos se han integrado, entonces, para permitir que la selección del modo de administración de memoria en SODIUM pueda realizarse mediante la utilización de un menú [6] que aparece luego de la carga del kernel en memoria, y antes de iniciar la interfaz de línea de comandos.

Se han logrado avances sobre otros aspectos del sistema operativo como, por ejemplo, la posibilidad de cambiar el modo de planificación, carga y vinculación de procesos e hilos en memoria.

Sin embargo, hasta el momento todas las integraciones reconfigurables hechas han sido realizadas *ad hoc*, luego de presentarse una implementación nueva de algún aspecto del sistema operativo y exigiendo realizar un trabajo de reingeniería para agregar la nueva alternativa, o nivel de separación conceptual entre ellas.

Debido a esto, surge la necesidad de diseñar y planificar una arquitectura que contemple *a priori* los aspectos que serán reconfigurables, y sus posibles implementaciones candidatas, y utilizarlo como mapa de ruta para futuros desarrollos y consignas de trabajos prácticos para los alumnos de la cátedra.

III. CONCEPTOS DE UN DISEÑO RECONFIGURABLE

A continuación presentaremos conceptos que utilizamos para evaluar el diseño de un sistema operativo reconfigurable.

A. Tiempos para la configuración

Los niveles temporales para la configuración se utilizarán para determinar en qué estado de ejecución del sistema operativo se podrá cambiar el modo de funcionamiento de un aspecto del mismo, y se pueden dividir en los siguientes niveles:

- **Nivel 1 - Configuración en Tiempo de Compilación:**
  En este nivel se sitúan los aspectos que son configurables únicamente vía código y pueden ser cambiados luego de recompilar el sistema operativo modificado, y ejecutando las rutinas de arranque. A diferencia de los siguientes niveles que contarán con todo el código y/o las bibliotecas necesarias para cada modo disponible, el sistema operativo contará con lo necesario para la implementación elegida en particular.

- **Nivel 2 - Configuración en Tiempo de Arranque:**
  En este nivel se sitúan los aspectos que son configurables mediante un menú o archivo de configuración. La configuración se establecerá antes de ceder la ejecución a la consola de línea de comandos o interfaz gráfica, y se define una única vez por sesión. Para cambiar el modo de funcionamiento para el aspecto en cuestión se deberá

reiniciar el sistema operativo y seleccionarlo.

- **Nivel 3 - Configuración en Tiempo de Ejecución Limitada:** En este nivel se sitúan los aspectos que permiten ser configurados de manera diferente para cada proceso, pero que no pueden ser reconfigurados para el mismo una vez que inició su ejecución.

- **Nivel 4 - Configuración en Tiempo de Ejecución Pura:** En este nivel se sitúan los aspectos que permiten ser configurados en cualquier momento, y aprovechados para y por un proceso durante la ejecución del mismo.

El diseño de una arquitectura reconfigurable debería intentar llevar al mayor nivel temporal posible cada una de las transiciones entre modos de los aspectos del sistema operativo dado que menos elementos del contexto se verán afectados por la reconfiguración.

Cualquiera de los niveles citados anteriores, a partir del nivel 2, son además considerados modificables si permiten a un usuario administrador agregar, remover o modificar modos de funcionamiento para el aspecto evaluado. Tal es el caso de muchos sistemas operativos tipo Linux [7] cuyo código de kernel y bibliotecas de usuario pueden ser modificados, recompilados y puestos en marcha, en diferentes niveles temporales, según el aspecto a modificar.

B. Transiciones entre modos en tiempo de ejecución

Llamaremos transiciones a cada cambio entre un modo de funcionamiento a otro diferente de un aspecto del sistema operativo.

Como se ha dicho, estas transiciones deberían minimizar consecuencias secundarias sobre tareas, recursos, o dispositivos que vayan más allá de la naturaleza del cambio.

Por ejemplo, el cambio de un modo de administración de memoria particionado a paginado podría realizarse de manera transparente, sin embargo, no es posible garantizar que el cambio inverso sea compatible; podría haber más espacio asignado a páginas en espacio virtual que no cabrían en un contexto particionado de solo memoria física. Sin embargo, la transición es tomada en cuenta ya que nos permitiría llegar a la conclusión de la gravedad de la problemática presentada.

Es por esto que el diseño debe contemplar también cada una de las transiciones entre modos para cada aspecto, ya que no todos son compatibles entre sí. Esto puede ser representado mediante un grafo dirigido [ver Figura 2] en el cual la presencia de una flecha uniforme indica una transición compatible (transparente), una flecha intermitente indica una transición con posible pérdida de información o desperdicio de recursos, y una flecha intermitente tachada indica una transición incompatible.

Nótese que se han omitido las relaciones transitivas perfectamente compatibles, por ejemplo, pasar de “modo de particiones equitativas” a “modo paginado con swapping”, ya que, al tratarse en definitiva del diseño de un producto software, contemplar todas las transiciones posibles —en definitiva, casos particulares— puede aumentar las líneas de código, tiempo de desarrollo, y dificulta su mantenimiento.

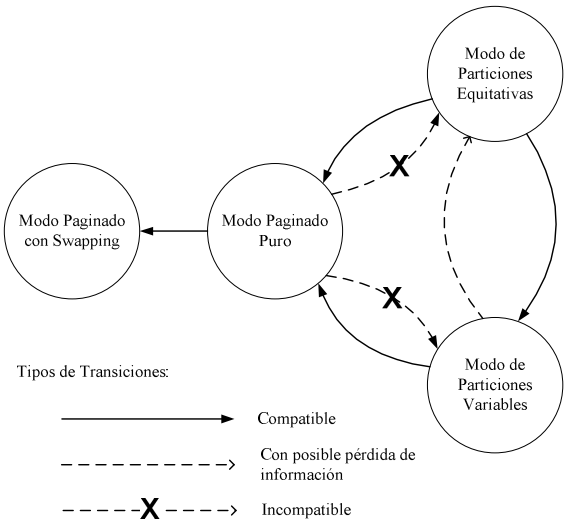

Figura 2. Ejemplo de diagrama de transiciones entre modos de administración de memoria en tiempo de ejecución

Por lo tanto, dada la necesidad de reducir el número de transiciones sin perder caminos posibles, resulta válido eliminar algunas relaciones transitivas y realizar los cambios de manera escalonada.

En el caso del ejemplo, es posible definir una jerarquía de modo que el pasaje de cualquier modo de particiones al modo de paginación pura funciona como primer paso hacia el pasaje a modo paginado con swapping.

Sin embargo, no resulta conveniente eliminar todas las relaciones transitivas ya que algunas representan cambios entre estados de un mismo nivel jerárquico (por ejemplo, entre los modos de particiones equitativas y variables), y no existe un orden lógico que indique desde cuál de ellas debe pasarse al siguiente.

Por un lado, Las transiciones incompatibles o de pérdidas irrecuperables difícilmente permitan superar el nivel temporal 2, por lo que sería válido dejar esas opciones como configurables en tiempo de arranque (no se incluyen en el diagrama). Por otro lado, Las transiciones con posible pérdida de información o desperdicio de recursos pueden llegar a realizarse hasta el nivel temporal 4, sin embargo, deben implementarse con verificaciones que avisen al usuario de los posibles conflictos que puedan surgir del cambio.

C. Facilidad de configuración

Dado que estamos evaluando el diseño de una arquitectura reconfigurable para fines didácticos, las interfaces que permitan el cambio entre implementaciones de los aspectos del sistema operativo deben ser fáciles de comprender y utilizar ya que estas no forman parte del sistema en sí, sino que son el medio por el cual efectuar los cambios.

En los sistemas basados en Unix, la mayor parte de la configuración se realiza a través de archivos de configuración [8] lo cual facilita la creación de scripts que automaticen gran parte del trabajo de administración.

Sin embargo, la utilización de archivos de configuración debería, en el caso de un sistema operativo didáctico, servir únicamente como medio de almacenamiento de la configuración, y no como interfaz para modificarla.

El diseño del sistema operativo didáctico debe contar con interfaces de intuitiva interacción como por ejemplo menús gráficos o variables de entorno para cambiar el modo de funcionamiento de alguno de sus aspectos.

D. Pasos para realizar el diseño

Finalmente, para realizar el diseño deben seguirse los siguientes pasos:

- Analizar la situación actual para cada transición —si es que existe alguna—, incluyendo factibilidad, interfaz por la cual se puede realizar, y nivel temporal.
- Enumerar los modos posibles y efectuar el análisis gráfico de los modos disponibles para descubrir nuevas transiciones posibles durante la ejecución.
- Realizar el diseño reconfigurable indicando detalles de implementación, nivel temporal, y elementos a verificar por posibles pérdidas de información para cada transición.
- Diseñar las interfaces de usuario que mejor se adecuen a la facilidad de configuración del caso.

IV. Ejemplo de Uso: Diseño de un Administrador de Memoria Reconfigurable

Para aplicar este método de diseño sobre el administrador de memoria de SODIUM primero hemos limitado los alcances de este aspecto a la administración del espacio disponible para ubicación de los segmentos de proceso y otros recursos de sistema en memoria ─principal y/o secundaria─, independientemente del modo de funcionamiento (la posibilidad de usar segmentación, el enlazado dinámico, y los modos de carga se verán en la siguiente sección).

Existe sobrada bibliografía sobre los diferentes modos de administración de memoria [9] [10] por lo que nos ocuparemos directamente de evaluar las transiciones entre ellos y los pasos para lograr el mayor nivel temporal para cada caso.

A. Situación actual

Actualmente, SODIUM permite la elección de cualquiera de los modos de memoria en la fig. 2, con un menú interactivo durante el arranque del sistema en el que el usuario puede elegir el deseado utilizando las flechas y la tecla de retorno del teclado.

Luego de efectuada la selección, no es posible realizar un cambio de modo sin reiniciar el sistema, por lo que todas las transiciones son de nivel temporal 2 (no tendría sentido diagramar este caso, ya que no habrían flechas a graficar).

B. Modos y transiciones posibles en tiempo de ejecución

Dado que SODIUM funciona en modo protegido y multiprogramación, no se contempla el uso de un único segmento de memoria, ya que éste solo aplica para sistemas operativos monoprogramados. Los modos disponibles para la administración de memoria en SODIUM son entonces:

- Modo de Particiones Equitativas (PE)
- Modo de Particiones Variables (PV)
- Modo Paginado Puro (PgP)
- Modo Paginado con Swapping (PgS)

El gráfico de transiciones está presentado en la Figura 3, por lo que las transiciones posibles son:

- PE → PV
- PE → PgP
- PV → ~PE
- PV → PgP
- PgP → PgS

C. Diseño reconfigurable

Como hemos indicado en los pasos para realizar el diseño, en este punto analizaremos cada transición para revisar la implementación, nivel temporal y posibles pérdidas para cada transición.

PE → PV: La transición desde el modo de particiones equitativas a la utilización de particiones variables debe realizarse creando tantas particiones variables como equitativas ocupadas existían al momento del cambio, y asignarles el tamaño de estas últimas.

También puede reducirse las particiones al tamaño del proceso que se encuentre en ella indicando fehacientemente el total de la memoria recuperada (de la fragmentación interna).

Una vez efectuada la conversión, tanto el posible espacio no asignado al final como las particiones libres contiguas pueden pasar a ser espacio libre para la asignación de nuevas particiones de tamaño variable [ver Figura 3].

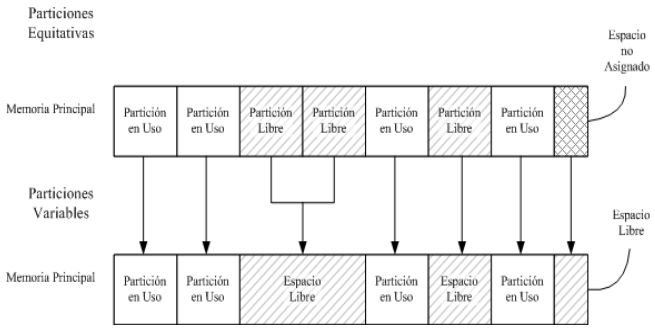

Figura 3. Ejemplo de conversión de modo de particiones equitativas a variables

Podrían existir otras formas de cambiar de modo que, al mismo tiempo eliminen la fragmentación de memoria y resuelvan el espacio libre directamente, pero ésta efectúa un cambio transparente ya que garantiza que las particiones queden en la ubicación física original.

Esta conversión no causa pérdidas de información ni desperdicio de recursos y puede ser realizada en cualquier momento mediante un llamado a sistema, y no afecta al funcionamiento del proceso, por lo que el nivel temporal es 3.

PV → ~PE: Para la transición desde el modo de particiones variables al de particiones equitativas se debe tomar el tamaño de la mayor partición existente como nuevo tamaño fijo

El resultado de este pasaje puede provocar que existan espacios no asignables si el tamaño de la partición más grande no es un divisor natural del tamaño total de la memoria [ver Figura 4].

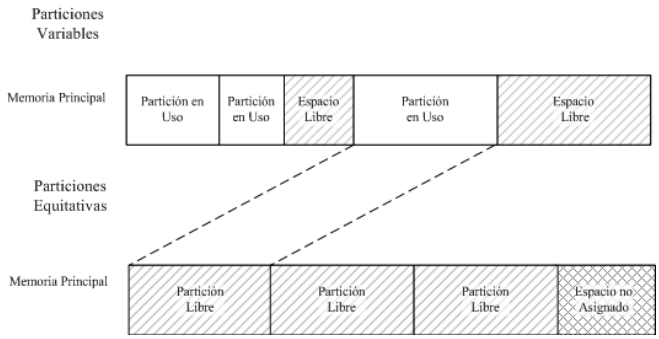

Figura 4. Ejemplo de conversión de modo de particiones variables a equitativas

El número total de particiones final será el resultado entero de la división entre el tamaño total de la memoria disponible por el tamaño fijo obtenido [ver Figura 5].

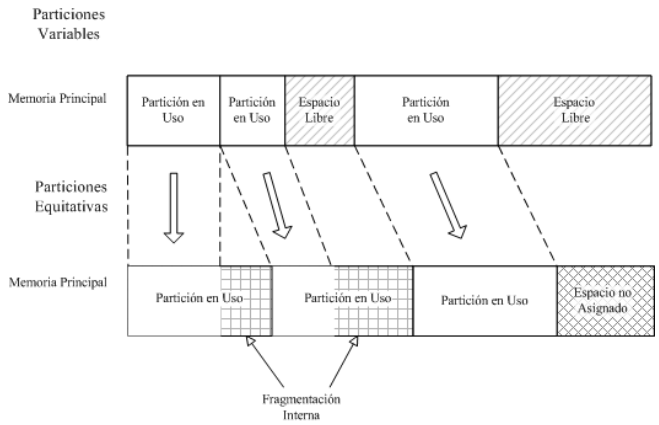

Figura 5. En la asignación de particiones puede producirse fragmentación interna

Como se puede observar en la figura 5, si existieran más de una partición y fueran de distinto tamaño, al efectuar la copia de memoria hacia su partición equitativa de destino, existirá una parte de ella que quedará desperdiciada para todos los casos menos uno, el de la partición más grande.

En el caso de que la cantidad de particiones equitativas resultantes fuera menor al de las particiones variables existentes, se estaría incurriendo en una pérdida de información ya que alguna de las particiones deberá ser descartada.

A pesar de causar un potencial desperdicio de recursos y/o pérdida de información, este cambio se puede implementar con un nivel temporal 3 utilizando mecanismos e interfaces que contemplen todos los casos.

PE → PgP y PV → PgP: El análisis de la transición particiones a modo paginado puro es similar para ambos casos por lo que podremos unificar el análisis tomando particiones equitativas como punto de partida.

Al activar el uso de particiones, cada proceso contará con un espacio virtual similar al espacio físico disponible de memoria principal usado en particiones.

Este espacio virtual es privado de cada proceso por lo que cada uno tendrá solamente una partición que podría conservar la misma ubicación absoluta —esto no es estrictamente necesario—.

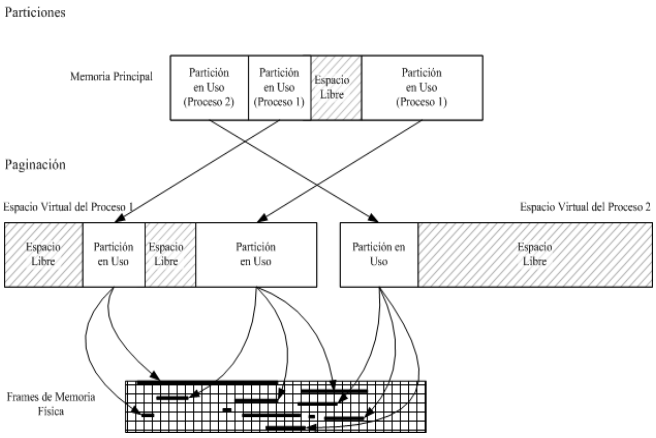

Figura 6. Ejemplo de conversión de modo de particiones variables a equitativas

Las áreas libres en los espacios virtuales de cada proceso no representan un desperdicio, a pesar de ser mayor ahora que cuando se utilizaba particiones, ya que esto no se traduce a un desperdicio a nivel físico; las páginas libres no están aún asignadas a frames —porciones de espacio del mismo tamaño que una página— de memoria [ver Figura 6].

Como la activación de paginación siempre tiende a ampliar el espacio virtual disponible no puede haber pérdida de información en estas transiciones. El espacio físico disponible se conserva, teóricamente, durante la transición aunque podría reducirse dado el tamaño de los directorios de páginas por proceso; sin embargo, si el número de particiones por proceso es muy elevado, podría mantenerse.

La transición podría realizarse en cualquier momento, sin embargo, no será de utilidad sino hasta que se instancie un nuevo proceso, por lo que el máximo nivel temporal es de 3.

PgP → PgS: La activación del uso de swapping en paginación habilita a la ampliación del espacio físico disponible (utilizando memoria secundaria) para la creación de más espacios virtuales, sin embargo, los espacios virtuales se mantienen del mismo tamaño.

Para realizar esto, se deben reemplazar los mecanismos de asignación de páginas para permitir definirlas como *no presentes* en memoria principal, habilitar las excepciones de falta de página, y los mecanismos que leen/escriben los frames correspondientes del soporte externo [ver Figura 7].

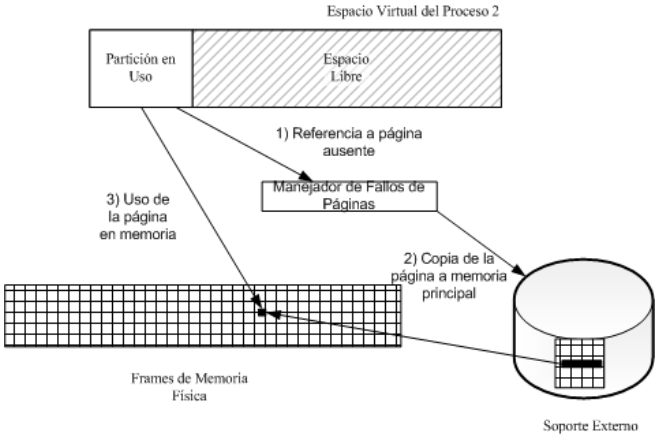

Figura 7. Pasos a implementar para activar paginación con swapping

Esta transición no causa pérdidas de información ni desperdicia recursos y afecta el transcurso de la ejecución de los procesos ampliando la memoria física disponible, y afectando el rendimiento por los accesos a dispositivos físicos por lo que el nivel temporal es 4.

Para el resto de las transiciones se puede mantener el nivel temporal 2 debiéndose reiniciar para realizar el cambio de modo.

Transiciones Incompatibles: El análisis de un cambio de modo de paginación a uno particionado resultó en una pérdida segura de información y en una posterior inestabilidad del sistema en el transcurso del pasaje. Es útil, sin embargo, dejar estas posibilidades contempladas en las interfaces, ya que significa un valor didáctico al alumno provocar la inestabilidad e investigar sus causas.

D. Interfaces para el cambio

Puede mantenerse el menú interactivo durante el proceso de arranque para la elección inicial de cualquiera de los modos. Dado que SODIUM no cuenta aún con una interfaz gráfica, puede implementarse una interfaz de texto auxiliar independiente de la consola de comandos en la que se puede cambiar el modo de memoria y otros aspectos del sistema utilizando los botones de función.

Esta interfaz debe informar y solicitar confirmación en caso de que la transición PV → ~PE pueda implicar pérdidas de información, o desperdicio de recursos, y explicar las causas de esto. Además deberá contemplar las transiciones incompatibles o de nivel temporal igual o menor a 2, indicando la futura inestabilidad, pérdida de datos, las causas que llevan a esto, e indicar fuentes de información a las que el alumno pueda recurrir para entenderlas.

V. HACIA UN SISTEMA OPERATIVO ADAPTABLE

Si bien hasta aquí hemos pensado el diseño de un sistema operativo cuyos aspectos puedan ser cambiados en tiempo de ejecución de manera manual, esta metodología abre las puertas a que la configuración pueda ser realizada automáticamente, dependiendo de las necesidades de espacio, procesamiento, e interacción con los dispositivos conectados.

Las ventajas de un sistema operativo autoconfigurable es que en cada momento puede aprovechar el mejor rendimiento de aquellos modos de un aspecto que son menos compatibles pero de mayor rendimiento, y cambiar de modo si, habiendo variado la demanda del uso del mismo, se precisa utilizar un modo más compatible.

Dado que para cada modo diferente de un aspecto se precisan que ciertas condiciones sean cumplidas, y que se ejecute una rutina al cambiarse de modo, hemos considerado que las herramientas de análisis que mejor sirven para describir este comportamiento son las *tablas de decisión* [11].

A. Usando una tabla de decisión estática

En el caso del ejemplo, podemos construir un autómata finito a partir del diagrama de transiciones utilizado para realizar el diseño reconfigurable [ver Figura 2] cuyos estados sean cada uno de los posibles modos del aspecto en cuestión a analizar, en este caso, de la administración de memoria.

Deberemos luego definir las condiciones necesarias a cumplir para cada uno de los modos recorriendo inversamente las transiciones incompatibles partiendo del modo más compatible (**PgS**) en dirección inversa hacia los nodos menos compatibles (**PE** y **PV**).

La necesidad de retornar a los modos menos compatibles es que, tanto en el caso de la administración de memoria como en otros aspectos del sistema operativo, la compatibilidad implica complejidades que pueden afectar al rendimiento, o a la facilidad de administración.

Las transiciones entre los estados deberán contemplar los siguientes aspectos:

- Las condiciones necesarias que, dado un cambio en el contexto, pasen a (o dejen de) ser cumplidas, para realizar la transición.
- Las acciones necesarias para pasar de un modo a otro representarán un mensaje emitido.
- Por simplicidad en el ejemplo, los datos de entrada (input) serán únicamente los sucesos de proceso saliente **PrS** (liberación de memoria), y proceso entrante **PrE** (requerimiento de memoria).

Nótese que en el autómata resultante [ver Figura 8], puede darse el caso en que las transiciones que, al realizarse de modo manual —dada la incapacidad del usuario de tener en cuenta todas las variables internas del sistema—, eran de nivel 2 ahora pueden ser realizados en niveles 3 o 4, dado que el sistema puede detectar automáticamente las condiciones específicas en las que no se incurrirá en pérdida de datos.

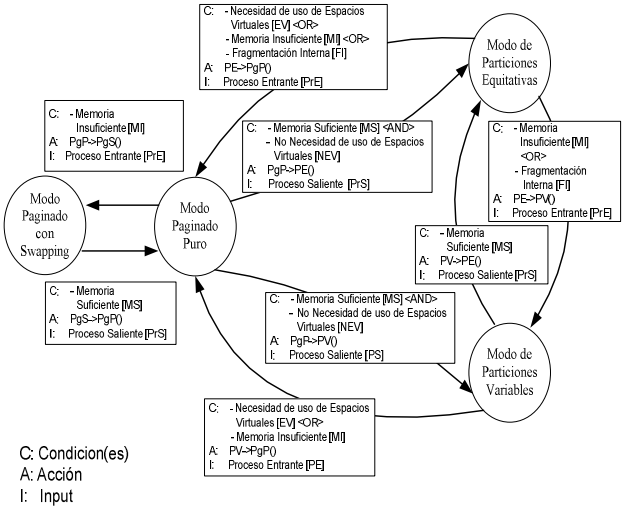

Figura 8. Autómata para cambios automáticos de modo de administración de memoria

Nótese también que, por lo general, las transiciones paralelas y opuestas, es decir, que van y vienen de los mismos dos estados, en sentido contrario, cuentan, en el mismo sentido, con condiciones opuestas. Por ejemplo, aquellas transiciones que se den por falta de memoria, podrán regresar al estado anterior solo si existe memoria suficiente.

Si bien los inputs y estados son también condiciones, la separación hecha responde a agregarle claridad a la aplicación de estas tablas a la visión de sistemas operativos.

El aporte didáctico de experimentar con una configuración automática será de mucha utilidad al indicar mediante mensajes a pantalla el preciso instante en el que se ha efectuado una transición y las condiciones que la propiciaron.

La decisión sobre cuál será la regla inicial estará dada por una elección de nivel temporal 2, ya sea por elección manual o archivo de configuración. En la práctica no existirá más estado final que aquel en el que se encuentre el sistema al apagarlo.

En la tabla de decisión obtenida [ver Figura 9], puede observarse que para aquellas transiciones con condiciones conectadas por disyunción lógica (OR), existen reglas con la misma acción; esto no significa que se produzca un indeterminismo, sino que diferentes causas pueden provocar la acción.

El indeterminismo sí se produce en las reglas que devuelven de **PgP** a **PV** o **PE** dado que el mismo conjunto de condiciones pueden llevar a efectuar cualquiera de las acciones, o en el caso de estar en **PE** y se de la condición de memoria insuficiente ante un proceso entrante.

En estos casos, se pueden probar las n-acciones posibles y evaluar si alguna de ellas sigue sin satisfacer las necesidades de memoria del proceso entrante.

Regla N°		1	2	3	4	5	6	7	8	9	10	11	12
Condiciones	MI	x				x					x	x	
	MS			x	x				x				x
	FI		x				x						
	EV							x		x			
	NEV				x				x				
Inputs	PrE	x	x			x	x	x		x	x		x
	PrS			x	x				x			x	
Estado	PE	x	x			x	x	x					
	PV			x						x			
	PgP				x				x				x
	PgS										x	x	
Acciones	PE->PV()	x	x										
	PV->PE()			x									
	PgP->PE()				x								
	PE->PgP()					x	x	x					
	PgP->PV()								x				
	PV->PgP()									x			
	PgS->PgP()										x	x	
	PgP->PgS()												x

Figura 9. Tabla de decisión estática para el ejemplo simplificado de administración de modos de memoria[1]

B. Usando una tabla de decisión adaptable

El caso del ejemplo, con cuatro estados y sus transiciones, es una simplificación a fines prácticos de presentar el método propuesto, de todo el mapa posible de estados y transiciones posibles para los modos de administración de memoria que puede haber, y habrá en SODIUM.

El problema de utilizar una tabla de decisión estática es que se parte de la hipótesis de que existe la(s) decisión(es) tomadas para cada combinación de condiciones serán siempre las más convenientes y, por lo tanto, se pueden definir *a priori*.

Sin embargo, durante la ejecución de un sistema operativo, pueden darse distintas opciones posibles de cambio de modo para un aspecto, o conjuntos de cambios de diferentes aspectos a la vez que den respuesta a una necesidad única y particular planteada por el propio uso del mismo.

La complejidad de un mapa de transiciones de modos resultante tal hace que sea no solo impracticable sino también ineficiente contemplar todas las combinaciones de condiciones-input que puedan darse.

Una alternativa que puede dar solución a un sistema de tal complejidad es el uso de las *tablas de decisión adaptables* [11] [12]. Tal como se describen en [11], las tablas de decisión adaptables son una extensión de las tablas de decisión convencionales con el agregado de meta-acciones —similares en el sentido estricto a las acciones convencionales— que pueden ser ejecutadas antes o después de las existentes, y cuyo fin es modificar, es decir, agregar, modificar, o eliminar reglas, la misma tabla para adaptar el comportamiento del

[1] MI – Memoria Insuficiente; MS – Memoria Suficiente; FI – Fragmentación Interna; EV – Necesidad de Espacios Virtuales; NEV – No Necesidad de Espacios Virtuales

sistema en base a los eventos que vayan ocurriendo y el contexto dado.

La elaboración de la tabla de decisión para un sistema operativo adaptable —y SODIUM, en el futuro— contará con las siguientes premisas:

- Pueden definirse acciones por defecto a tomar en cada regla mediante el método definido en la Sección III, sin evaluar casos específicos, excepciones, ni relaciones con otros aspectos del sistema operativo.
- Se definirán acciones adaptativas de ejecución previa que evalúen la compatibilidad de las reglas de transición actuales, elimine aquellas que no sean compatibles con el estado actual del sistema, y agregue aquellos estados que sí lo son.
- Otra acción adaptativa de ejecución previa determinará en base a un sistema de métricas la acción adecuada en cada momento ante cada evento de input.
- Se definirán acciones adaptativas de ejecución posterior para la toma de métricas sobre transición posible (rendimiento de ejecución, costo de transición, frecuencia de cambio, frecuencia de compatibilidad, etc.).
- El estado de la tabla se guarda antes de apagar el sistema y se toma como tabla inicial en la siguiente sesión.

El esquema de métricas para la determinación de las mejores reglas es un campo que merece la pena ser evaluado en posteriores estudios en el camino hacia un sistema operativo adaptable.

Las premisas para la creación de la tabla de decisión adaptable determinarán la capacidad de aprendizaje del sistema operativo para determinar cuáles serán las mejores transiciones en función de cómo es usado. Esto implicará que dos usuarios con necesidades de uso diferentes contarán, luego de un lapso de uso razonable, con una configuración del sistema operativo personalizada que se adapta de manera óptima y diferente para cada caso.

VI. Conclusión

Mediante el método expuesto, hemos podido contemplar el diseño reconfigurable de un aspecto de un sistema operativo didáctico que hasta el día de hoy, si bien puede ser cambiado luego de reiniciar el sistema, no permite ser cambiado mientras el sistema se encuentra en ejecución.

Este enfoque nos permitirá efectuar el diseño de la reconfigurable para el resto de todos los aspectos de SODIUM que aún permanecen definidos por compilación y que serían de mayor valor didáctico si pudieran cambiarse durante la ejecución.

El administrador de memoria ha sido el primer aspecto reconfigurable en el que hemos aplicado esta metodología, obteniendo como resultado una mayor comprensión *a priori* Mejorando el entendimiento —tanto de docentes como de alumnos— de los mecanismos e interfaces respecto al enfoque *ah hoc* utilizado otrora.

La combinación de este enfoque, con las tecnologías de dispositivos adaptables existentes nos acerca la posibilidad de obtener un sistema operativo que responda de manera diferente acorde a las necesidades de uso de distintos usuarios.

El potencial práctico y didáctico del desarrollo por parte de profesores y alumnos de un sistema operativo adaptable es, de por sí, motivo suficiente para continuar tanto con la investigación en este sentido, como con la aplicación práctica sobre SODIUM de los avances logrados.

Referencias

[1] J. C. Dvorak, *Vista's 11 Pillars of Failure*, PCMag.com, 2008.

[2] Steven J. Vaughan-Nichols, "The 10 Worst Operating Systems of All Time", *PC World Magazine*, 2009.

[3] A. C. Veitch; N. C. Hutchinson, "Kea-a dynamically extensible and configurable operating system kernel", Configurable Distributed Systems, 1996. Proceedings., Third International Conference on.

[4] Backwards Compatibility – Microsoft Versus, http://www.msversus.org/backwards-compatibility.html

[5] W. Stallings, *Sistemas Operativos*, Capítulo 7: "Gestión de Memoria", Prentice Hall, 4ª Edición, 2001 .

[6] N. Casas; G. De Luca; S. Martin; H. Ryckeboer; M. Cortina; G. Puyo; W. Valiente, "Algoritmo de Administración de Memoria en un Sistema Operativo Didáctico", *Anuario de Investigaciones, Resúmenes extendidos 2009*, Universidad Nacional de La Matanza, ISBN: 978-987-1635-24-8.

[7] Kurt Wall et ál, *Programación en Linux Al descubierto*, Prentice Hall, 2ª Edición, 2001.

[8] A. S. Tanenbaum, *Sistemas Operativos Modernos*, Prentice Hall, 1ª Edición, 1993.

[9] Mark E. Russinovich; D. A. Solomon, *Microsoft Windows Internals*, Microsoff Press, Fourth Edition, 2005, págs. 251-255 y 615-654.

[10] A. S. Tanenbaum; A. S. Woodhull, *Operating Systems, Design and Implementation*, Prentice Hall, 2ª Edición, 1997.

[11] J. J. Neto, "Adaptive Rule-Driven Devices - General Formulation and Case Study" Lecture Notes in Computer Science, 2002, Volume 2494/2002. Springer

[12] T. Pedrazzi, A. Tchemra, R. Rocha "Adaptive Decision Tables A Case Study of their Application to Decision-Taking Problems" Adaptive and Natural Computing Algorithms 2005, Part III. Springer.

Sergio Miguel Martin se graduó en la carrera de Ingeniería en Informática en la Universidad Nacional de La Matanza (UNLaM) de Buenos Aires, Argentina el año 2010. Es docente ayudante de las materias de Sistemas Operativos desde el 2010; de Autómatas y Lenguajes Formales desde el 2011; y de Métricas de Software desde el 2011. Actualmente se encuentra cursando la Maestría en Ingeniería en Informática dictada en la Escuela de Posgrado de UNLaM. Sus campos de investigación se centran en el diseño y la enseñanza de Sistemas Operativos.

Graciela Elisabeth De Luca nació en la ciudad de Buenos Aires, Argentina, 13 de Junio de 1959, título Analista Universitaria de Sistemas de La Universidad Tecnológica Nacional y Licenciada en Informática de la Universidad Católica de Salta. Desde el año 2005 pertenece al grupo de Investigación de la Universidad Nacional de la Matanza, en el área de Arquitectura y Sistemas Operativos. Trabajo de tesis en curso de la maestría en Informática de la Universidad Nacional de la Matanza.

Nicanor Blas Casas, nació en la ciudad de Buenos Aires, Argentina. Título Analista Universitario de Sistemas de la Universidad Tecnológica Nacional, Ingeniero en Informática de la universidad Católica de Salta. Desde el año 2005 pertenece al grupo de Investigaciónde la Universidad Nacional de la Matanza, en el área de Arquitectura y Sistemas Operativos. Trabajo de tesis en curso de la maestría en Informática de la Universidad Nacional de la Matanza.