Encuentro Internacional de Educación en Ingeniería ACOFI
Una formación de calidad en ingeniería para el futuro
Centro de Convenciones Cartagena de Indias
15 al 18 de Septiembre de 2015

ENSEÑANZA DE SISTEMAS OPERATIVOS CON UN SIMULADOR DIDÁCTICO FÁCILMENTE EXTENSIBLE

Helmuth Trefftz Gómez, Juan Francisco Cardona Mc'Cormick

Universidad EAFIT
Medellín, Colombia

Resumen

Sistemas operativos es una de las asignaturas más importantes de programas como Ingeniería de Sistemas o Ingeniería Electrónica. La relación entre la teoría y la práctica, al momento de enseñar Sistemas Operativos, cambia de un enfoque a otro. Los estudiantes que van a implementar sistemas embebidos se pueden beneficiar de un enfoque más práctico, en el cual con frecuencia se estudia el código fuente de un sistema operativo real. Este proceso puede resultar tedioso para algunos estudiantes. Por el contrario, un enfoque más teórico es apropiado para estudiantes que en su vida profesional van a utilizar los sistemas operativos. En este artículo se presenta una metodología novedosa que constituye un punto intermedio, permitiendo que los estudiantes desarrollen ciertos componentes del sistema operativo partiendo de un simulador de un sistema operativo muy sencillo. Consideramos que esta metodología se puede extrapolar a un número importante de asignaturas, tanto en ingenierías directamente relacionadas con la computación como en otros campos de la disciplina.

Palabras clave: sistemas operativos; simulación; enseñanza de la ingeniería

Abstract

Operating Systems (OS) is one of the most important subjects in Computer Engineering or Electrical Engineering. The ratio between theory and practice, when teaching OS, can vary depending on the teaching approach. Students, who will implement embedded systems in their professional lives, may benefit from a more practical approach, in which the source code of a real OS may be studied. This approach, nevertheless, may be tedious for many students. On the other hand, a more theoretical approach is appropriate for students who will only use the characteristics of the OS. In this paper, a novel methodology is presented that embodies an intermediate

approach, allowing students to implement certain components of an OS using a simulator of a very simple system as starting point. We consider that this methodology can be easily extrapolated to an important number of subjects, both in computer-related Engineering fields and other non-computer related fields.

***Keywords**: operating systems; simulation; engineering teaching*

1. Introducción

Sistemas Operativos es una de las asignaturas más importantes en los planes de estudio de los programas de computación y electrónica. En vista de que el Sistema Operativo es el programa que coordina el uso eficiente de los recursos de un computador, es muy importante que los estudiantes de dichas carreras conozcan su funcionamiento. Sin embargo, el enfoque para la enseñanza de Sistemas Operativos cubre un espectro muy amplio. En un extremo está el enfoque que prepara a los estudiantes para *programar* parte de un sistema operativo. Este enfoque es el utilizado para preparar profesionales que, por ejemplo, desarrollan sistemas embebidos. En estos cursos con frecuencia se estudia el código fuente de algunos sistemas operativos, lo cual puede ser complejo y tedioso. En el otro extremo, está el enfoque que prepara a los estudiantes para *utilizar* los sistemas operativos razonablemente bien. Este es el enfoque que se usa tradicionalmente para preparar profesionales que, por ejemplo no desarrollan software o lo hacen de manera eventual. En estos cursos hay un componente grande de teoría que se complementa con unas prácticas de baja complejidad.

En este documento se presentan los resultados de un enfoque novedoso intermedio que combina las ventajas de los enfoques mencionados anteriormente. Se reporta la utilización de un simulador, desarrollado específicamente para el curso, en el cual se simulan aspectos de hardware (procesador, registros, memoria) y software (ensamblador, "*loader*" y "*scheduler*"). El estudiante dispone del código fuente del simulador, que se escribe buscando dos objetivos: (i) *sencillez*, para que sea fácilmente entendible para los estudiantes y (ii) *extensibilidad*, para que los estudiantes puedan implementar fácilmente los proyectos que se proponen.

A medida que el curso avanza, se entrega una nueva versión del simulador, en la cual se cuenta con una implementación básica del tema a cubrir. Por ejemplo, en el tema de "scheduling", se entrega un "scheduler" básico funcionando y los estudiantes deben implementar diferentes algoritmos, obtener estadísticas del desempeño y explicar las razones de los cambios en el desempeño. En el tema de manejo de memoria, se entrega una versión del simulador que implementa un algoritmo básico de manejo de páginas y se solicita a los estudiantes que implementen diferentes algoritmos de manejo de páginas y que, de nuevo, obtengan estadísticas y expliquen los resultados. En todos los casos, los estudiantes realizan presentaciones a la clase con los resultados de sus proyectos.

El resto de este artículo está organizado de la siguiente manera: En la sección 2 se presenta una revisión de propuestas en torno a la enseñanza de Sistemas Operativos. En la sección 3 se describe el simulador básico que se utiliza como punto de partida para el curso. En la sección 4 se describen las diferentes extensiones que los estudiantes deben realizar al simulador del curso. En la sección 5 se discuten los resultados del curso. Finalmente, en la sección 6 se presentan las conclusiones y posibles trabajos futuros.

2. Trabajos Relacionados

La práctica en las asignaturas de Sistemas Operativos es un componente muy importante. Por un lado, un grupo importante de profesores proponen realizar modificaciones al código fuente de Sistemas Operativos reales. De esta manera los estudiantes pueden conocer a fondo un Sistema Operativo y conocer el estilo de programación utilizado por sus programadores. Sin embargo, este enfoque supone que el estudiante comprenda un programa extremadamente grande y por eso difícil de entender. Adicionalmente, es común que los cambios realizados estropeen el sistema operativo y por esto el computador que se utiliza para las pruebas posiblemente no pueda ni siquiera arrancar. Nieh y Vail (1), profesores de la Universidad de Columbia, reportan el uso de una plataforma virtual sobre la cual los estudiantes hacen pruebas de una versión del sistema Linux. El utilizar una plataforma virtual hace que las consecuencias al introducir un error en el sistema sean menos graves, pero el trabajar con el código del Sistema Operativo Linux es, de todas formas, muy complejo.

El Sistema MINIX fue diseñado con la idea de se un sistema sencillo para ser utilizado para enseñar Sistemas Operativos, tal como lo expresa Andrew Tanenbaum, su creador, en artículos que datan desde 1987 (2). De acuerdo con su creador, el MINIX fue un precursor directo del LINUX, uno de los sistemas operativos más utilizados en la actualidad. Sin embargo, el MINIX ha crecido en el tiempo y es, hoy en día, un programa de un tamaño tal que no es factible estudiarlo en su totalidad en un curso. Con un objetivo similar, se han creado sistemas operativos tratando de limitar su complejidad. Este es el enfoque seguido por Anderson et. al. (3) con el sistema NACHOS, que aún se utiliza en algunas universidades. NACHOS ha contado con diversas ramificaciones, como PINTOS (4) de la Universidad de Stanford o OS/161 (5) de la universidad de Harvard.

Por otro lado, un gran número de profesores en el mundo no buscan que sus estudiantes escriban el código de un sistema operativo, sino que sepan utilizar las rutinas que se exponen por medio de un API. Hill *et. al.*, de la academia militar WestPoint, por ejemplo, proponen formas novedosas de explicar los conceptos de la administración de sistemas operativos utilizando juegos y acertijos (6). Los autores reportan que los estudiantes que utilizaron algunos de los juegos tuvieron un mejor desempeño en los exámenes que evaluaron los conocimientos explicados a través de los mismos.

3. El Simulador Básico

El simulador que se utiliza como punto de partida para el curso consta de dos programas: (i) el ensamblador y (ii) el simulador en si. Todos los componentes del simulador se implementan en el lenguaje Java, por facilidad para los estudiantes.

El ensamblador se basa en los capítulos 9 y 10 del libro *Compiler Construction* de Nilaus Wirth (7). El ensamblador toma como entrada código ensamblador de un lenguaje muy similar al MIPS y crea, como salida, código binario, de nuevo muy similar al código ejecutable del MIPS. De esta manera se crean 4 tipos de operaciones: Registro-registro (5 operaciones), registro-operador-inmediato (5 operaciones), registro-memoria (4 operaciones) e instrucciones de salto (9 operaciones). Uno de los objetivos principales del programa es la sencillez, buscando que el código sea fácilmente comprendido por los estudiantes. Por este motivo, no se incluyen todas las operaciones del lenguaje MIPS, solamente aquellas que son necearías para demostrar la ejecución de un programa. Adicionalmente se implementan 4 directivas de compilación: .data indica el comienzo de los datos, .text indica el comienzo del código, .space se utiliza para reservar espacio y .word se utiliza para definir el valor inicial de una variable.

El simulador de ejecución se compone de un cargador (implementado en la clase Loader.java) y un *scheduler* (implementado en la clase Scheduler.java). El cargador se encarga de poner los programas (tanto datos como código) en la memoria, separando un espacio para el manejo de la pila de ejecución al final del espacio de direcciones asignado al programa. El *scheduler* se encarga de simular la ejecución, instrucción por instrucción, de los programas que han sido cargados en memoria.

Se implementa tanto una instrucción de entrada de datos (trap 3) y una instrucción de salida (trap 5), ambos relacionados con la consola. De esta manera el estudiante puede verificar si el programa se ejecuta de manera correcta.

El simulador se documenta de manera extensiva. Las tablas 1, 2, 3 y 4, tomadas de la documentación del simulador, explican las instrucciones que se implementan en el simulador. El título de las tablas está en inglés porque el curso se ofreció en ese idioma.

Syntax	**Semantics**	**Example**
mov a,c	R.a = R.c	mov r1, r4
movn a,c	R.a = - R.c	movn r1, r4
add a,b,c	R.a = R.b + R.c	add r1, r4, r5
sub a,b,c	R.a = R.b - R.c	sub r1, r4, r5
cmp b,c	z = (R.b == R.c); n = (R.b < R.c)	cmp r4, r5

Tabla 1. Instrucciones Registro a Registro

Syntax	Semantics	Example
movi a,im	R.a = im	movi r1, 4
movni a,im	R.a = - im	movni r1, 4
addi a,b,im	R.a = R.b + im	addi r1, r4, 5
subi a,b,im	R.a = R.b - im	subi r1, r4, 5
cmpi b,im	z = (R.b == im); n = (R.b < im)	cmpi r4, 5

Tabla 2. Instrucciones registro a Operador Inmediato

Syntax	Semantics	Example
ldw a,im(b)	R.a = Mem[(BR + im + R.b)/4]	ldw r5,vec(r2)
pop a,b,im	R.a = Mem[R.b / 4]; R.b = R.b + im	pop r13,r14,4
stw im(b),a	Mem[(BR + im + R.b)/4] = R.a	stw vec(r2),r1
psh a,b,im	R.b = R.b - im; Mem[R.b / 4] = R.a	psh r13,r14,4

Tabla 3. Instrucciones Registro a Memoria

Syntax	Semantics	Example
beq disp	if(z) pc = pc + disp * 4	beq loop
bne disp	if(!z) pc = pc + disp * 4	bne loop
blt disp	if(n) pc = pc + disp * 4	blt loop
bge disp	if(!n) pc = pc + disp * 4	bge loop
ble disp	if(z *or* n) pc = pc + disp * 4	ble loop
bgt disp	if(!z *and* !n) pc = pc + disp * 4	bgt loop
br disp	pc = pc + disp * 4	br loop
bsr disp	R.14 = pc + 4; pc = pc + disp * 4;	bsr routine
ret	pc = R.14	ret

Tabla 4. Instrucciones de Salto

En la documentación se incluyen diversos ejemplos, con una complejidad creciente, para que los estudiantes se acostumbren a escribir código ensamblador y a estudiar su ejecución. Se incluyen instrucciones para manejo de pila en tiempo de ejecución, lo cual permite un manejo adecuado de funciones con paso de parámetros por valor, por referencia e, incluso, funciones recursivas.

4. Proyectos de curso y extensiones

En los primeros proyectos del curso se busca que los estudiantes se familiaricen con el lenguaje ensamblador y con el simulador. Para este propósito se les ofrecen, inicialmente, unos programas que deben modificar, y estudiar la ejecución en el simulador. Más tarde deben crear sus propios programas a partir de una especificación básica.

4.1. Algoritmos de *scheduling*

El simulador incluye un *scheduler* que asigna el procesador un número fijo de instrucciones a cada programa, y los va asignando de manera *round-robin*. En este proyecto, cada grupo de estudiantes debe modificar el algoritmo para implementar una forma diferente de asignación del procesador, con base en los algoritmos propuestos en el texto del curso (8). Los estudiantes deben proponer un conjunto de

programas (*mix*) normales y un conjunto de programas que demuestren las bondades del algoritmo de *scheduling* asignado. En cada caso deben tomar medidas de la ejecución. Cada grupo genera un reporte y expone sus resultados frente a sus compañeros de clase.

4.2. Ejecución de hilos concurrentes

El simulador se modifica para permitir la ejecución concurrente de hilos. Los hilos se simulan definiendo diferentes puntos de comienzo de la ejecución (main1, main2 y así sucesivamente). Los datos son compartidos, para poder simular condiciones de concurrencia. Inicialmente se muestra la ejecución de dos hilos que actualizan un contador compartido sin exclusión mutua. Se muestra que, de esta manera, el programa produce resultados equivocados. Para el manejo de la exclusión mutua se introduce una instrucción TSL (*test-and-set-lock*), la cual permite verificar el valor de una variable (*lock*) y asignarle un valor, ambas acciones de manera atómica. Los estudiantes deben modificar el programa para lograr exclusión mutua por medio de la instrucción TSL, verificando que el valor del contador sea correcto. Mientras un hilo tiene acceso a la variable compartida, el otro hilo debe realizar un ciclo, tratando de ganar acceso a dicha variable, haciendo una espera activa. Luego los estudiantes deben generar situaciones de abrazo mortal (deadlock), "olvidando" liberar la variable compartida.

4.3. Comunicación entre hilos mediante instrucciones señales

Para simular el envío de señales entre hilos concurrentes, el simulador se modifica adicionando la implementación de una instrucción para bloquear el hilo actual, similar a la instrucción Object.wait() del lenguaje Java, así como una instrucción que puede despertar otro hilo, similar a la instrucción Object.nofity() del mismo lenguaje. La instrucción para bloquear el hilo actual se implementa mediante una instrucción de Trap (trap 6) y la instrucción para despertar mediante otra instrucción de Trap (trap 7). Se modifica el simulador para que permita bloquear un hilo, así como el desbloqueo por parte de otro hilo. Utilizando de manera adecuada estas operaciones, en combinación con la instrucción TSL, los estudiantes deben implementar las instrucciones *up* y *down* de un semáforo con contador. De nuevo se prueba el funcionamiento correcto e incorrecto de un programa basado en hilos que actualizan el valor de una variable de manera concurrente.

4.4. Memoria virtual

Para demostrar el uso de memoria virtual, se modifica el simulador para que incluya dos niveles de memoria: memoria física y memoria virtual. Ambos tipos de memoria se implementan en el simulador utilizando arreglos, pero se utilizan consideraciones de tiempo para leer o escribir una página y determinar, de esta manera, el tiempo de ejecución total de los programas. El simulador se modifica incluyendo un algoritmo básico de memoria virtual. Como algoritmo de remplazo de páginas se utiliza el LRU, utilizando un contador de instrucciones ejecutadas por el simulador al momento de acceder a una posición de memoria que esté en la página correspondiente. La

simulación de las instrucciones relacionadas con la memoria (ldw, stw, pop, push, tsl) se modifica para verificar primero si la posición que se accede está en memoria física o no. Los estudiantes deben escribir un programa que suma los elementos almacenados en un arreglo de dos maneras: una que minimice el número de páginas a leer o escribir y otra que implique un número elevado de operaciones de páginas; se deben tomar tiempos de ejecución, comparar y explicar.

4.5. Algoritmos de remplazo de páginas

En este proyecto cada grupo debe implementar un algoritmo para reemplazo de páginas. Los estudiantes pueden modificar fácilmente elementos como el tamaño de la página, el tamaño de la memoria física, el tamaño de la memoria virtual, entre otros. De manera similar al proyecto de los algoritmos de *scheduling*, se deben crear programas que hagan un buen uso del algoritmo de remplazo de páginas escogido, así como uno que cause un número excesivo de operaciones de páginas. En cada caso se toman tiempos, se comparan los resultados y se explican éstos en un informe. De nuevo cada grupo realiza una presentación frente a la clase.

5. Resultados

El simulador se utilizó en uno de los grupos del curso de Sistemas Operativos de la Universidad EAFIT durante el primer semestre del 2015. Algunos de los proyectos más sencillos relacionados con el simulador, fueron realizados de manera individual. Los proyectos más complejos, en los cuales se requería realizar dos implementaciones diferentes, comparar los tiempos y realizar presentaciones, fueron realizados de manera grupal. Los proyectos individuales fueron bastante exitosos, con un promedio de nota de 4.2, 3.8 y 4.2. Los proyectos grupales, de una dificultad mayor, no fueron tan exitosos, teniendo un promedio de nota de 3.2. En el curso se realizaron también evaluaciones teóricas. En las evaluaciones teóricas relacionadas con los temas que se cubrieron en los proyectos de programación, tuvieron respuestas correctas por parte de la mayor parte de los estudiantes; lo cual parece sugerir que la programación de dichos conceptos permite una comprensión más profunda por parte de los estudiantes. Nuestra hipótesis es que, antes de programar una solución, el estudiante necesita tener una claridad meridiana acerca del concepto; de lo contrario no es posible realizar una programación adecuada.

En las encuestas de evaluación del curso los estudiantes mencionaron, de manera espontánea, que se habían sentido a gusto con la metodología basada en proyectos.

6. Conclusiones y Trabajo Futuro

Esta primera experiencia nos lleva a pensar que la metodología basada en extender el prototipo si constituye una metodología adecuada para explicar los conceptos de Sistemas Operativos. Para tener unas conclusiones más claras, pensamos realizar una experiencia comparando la metodología tradicional (con proyectos que utilicen la API

de un sistema operativo) con la metodología propuesta en torno al simulador, descrita en este artículo. Algunas preguntas de los exámenes serán comunes en ambos grupos, con miras a comparar los aprendizajes de ambos estudiantes.

En términos del simulador, pensamos extender la funcionalidad para incluir sistemas de archivos. Se entregaría un sistema de archivos básico (por ejemplo basado en i-nodes) y los estudiantes tendrían que implementar otro diferente (por ejemplo basado en *File Allocation Tables*).

7. Referencias

(1) Nieh, J. and Vaill, C. Experiences teaching operating systems using virtual platforms and Linux. In SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education. February 2005.
(2) Tanenbaum, A. A Unix Code with Source Code for Operating Systems Courses. In SIGOPS Operating Systems Review. Volume 21, issue 1. January 1987
(3) Wayne A. Christopher, Steven J. Procter, and Thomas E. Anderson. 1993. The Nachos instructional operating system. In Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings (USENIX'93).
(4) Ben Pfaff, Anthony Romano, and Godmar Back. 2009. The pintos instructional operating system kernel. SIGCSE Bull. 41, 1 (March 2009), 453-457.
(5) David A. Holland, Ada T. Lim, and Margo I. Seltzer. 2002. A new instructional operating system. SIGCSE Bull. 34, 1 (February 2002), 111-115.
(6) John M. D. Hill, Clark K. Ray, Jean R. S. Blair, and Curtis A. Carver, Jr.. 2003. Puzzles and games: addressing different learning styles in teaching operating systems concepts. In Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03). ACM, New York, NY, USA, 182-186.
(7) Wirth, Niklaus, et al. Compiler construction. Vol. 1. Reading: Addison-Wesley, 1996.
(8) Andrew S. Tanenbaum and Herbert Bos. 2014. *Modern Operating Systems* (4th ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.