Calendarizadores de procesos para PintOS

David Cordero Chavarría Manuel Zumbado Corrales
dacoch215@estudiantec.cr manu3193@gmail.com

Área de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—In the following paper, we present the challenges of implementing thread schedulers for a Computer Operating System, even a relatively small one like PintOS, in which the implementetion is focused and based on, some recommendations for programmers attempting to do similar projects, as well as the justification of the decisions made during design, and implementation of the several schedulers, and other necessary modules.**

***Palabras clave*—Operating Sistems, C Programming, PintOS, Thread Scheduling**

I. Introducción

Un conocimiento robusto del comportamiento de un sistema operativo, sólo se puede construir a través del diseño y la implementación de sistemas operativos, o por lo menos de partes de estos. En este proyecto se documenta la implementación de diferentes algoritmos de calendarización de procesos o hilos de ejecución del Sistema Operativo PintOS, un sistema mínimo creado en la universidad de Stanford para ilustrar principios de la implementación de Sistemas Operativos a Estudiantes de Ingeniería y Ciencias de la Computación.

II. Descripción del Problema

El rendimiento de un sistema operativo depende en gran parte su calendarizador de procesos, y del algoritmo que use para determinar qué proceso ejecutar, por lo que el éxito de un sistema operativo puede depender en gran parte del algoritmo de calendarización utilizado en su implementación.

Este proyecto se centra en la implementacón de diferentes algoritmos de calendarización para el Sistema Operativo PintOS, los cuales son: Round Robin, First Come First Served, Shortest Job First, y el calendarizador avanzado del Sistema Operativo BSD versión 4.4, que es un ejemplo de un Multi-Level Feedback Queue Scheduler.

Estos calendarizadores tienen como objetivo el manejo eficiente de los recursos de la computadora por parte del sistema operativo que los ejecuta para administrar procesos y evitar, mitigar, o eliminar los problemas asociados a ésta tarea, como la "starvation" de procesos, los deadlocks, etc.

III. Detalles de la implementación

El código fuente de PintOS carece de muchas características implementadas, aunque provee un calendarizador básico, sobre el cuál se basó la implementación de los algoritmos, el estudio de éste fue fundamental en el entendimiento de las funciones y bibliotecas proveídas en el código fuente, así como de las limitaciones de tamaño, tipos de datos y operaciones permitidas en el código.

El calendarizador avanzado, basado en BSD v4.4, usa hilos con prioridades, además de un campo adicional, llamado "nice'que indica qué tan "amable" es un hilo con otros hilos, y éstos se usan en el cálculo de la escogencia del siguiente hilo a ejecutar o más específicamente, en cómo ordenar los hilos en la cola, pues se usa una fórmula para ordenar los hilos en la cola al ordenarlos una vez que están por terminar su ejecución.

IV. Justificación

Dada cierta falta de familiaridad con sistemas de arranque, paso de opciones de arranque a un sistema operativo desde arranque de una computadora real, y limitaciones del código fuente de PintOS, al obtenerlo de la "tarball" del sitio web de Stanford, la creación de un sistema que arranque en una computadora física tiene el problema de que el sistema no encuentra "el disco duro" que para

PintOS se trata de un archivo binario, por lo que arroja un *kernel panic*

a como está descargado de Stanford, además de la carencia de tiempo para familiarizarse con el funcionamiento de todos los componentes presentes,

Para el calendarizador avanzado, se utilizaron las fórmulas descritas en la documentación de Stanford de PintOS, además de una lista de hilos para la implementación de la cola de prioridad, pues dado lo básico del sistema, y la naturaleza ilustrativa de los principios de sistemas operativos del proyecto, se consideró innecesario crear más de una cola, pues dada la baja cantidad de hilos a ejecutar se consideró que no se obtendría una mejora significativa de rendimiento al buscar elementos en varias colas, en vez de una sola, pues esta no estaría tan llena de elementos como para que los recorridos sobre ésta sean lentos.

V. Comparación entre los algoritmos

En cuanto a complejidad, el algoritmo de colas multinivel realimentadas es el msá complejo, inclusive en la implementación de éste sistema, que es sencillo en comparación a sistemas más grandes, como GNU/Linux, Solaris, o FreeBSD, requiere de bastantes cambios, además del uso de varios métricas para tomar la decisón de cuál hilo ejecutar y en qué orden, para obtener el mayor rendimiento posible, además de hacerlo evitando la "starvation".

El algoritmo First Come First Served es el conceptualmente más simple, además de ser una base para la implementación de los demás, que introducen algún nivel de complejidad extra.

El algoritmo Shortest Job First, básicamente agrega ordenamiento al First Come First served, lo que logra que trabajos que se ha determinado a priori que van a durar menos en terminar, logren hacerlo, reduciendo así el tiempo de espera promedio de todos los procesos, aunque no tomando en cuenta necesidades de procesamiento (un thread con un trabajo considerablemente importante, como un daemon que controle un dispositivo, podría tener que esperar a que varios procesos más cortos terminen, perdiendo así datos de entrada).

El algoritmo Round Robin, es un algoritmo justo, i.e. Garantiza que no existe la 'starvation', aunque introduce el problema de decisión de la cantidad de tiempo a darle a todos los hilos, lo que puede ser mitigado con un gran conocimiento a priori de los tipos de tareas a realizar en una máquina específica, pero ésto no es siempre posible por lo que se puede tratar de determinar un tiempo justo mediante la obtención de datos empíricos sobre el rendimiento de algunos tipos de procesos.

VI. Conclusiones

Una considerable dedicación de tiempo es necesaria para el desarrollo de sistemas operativos, lo mismo es cierto para PintOS, pues aun cierta familiaridad con el lenguaje de programación C no es suficiente para poder programar los diferentes elementos necesarios para la implementación de los calendarzadores realizados en éste proyecto, así como en otros posibles proyectos, como implementación de memoria virtual, sistemas de archivos, etc.

El conocimiento sobre algoritmos y conceptos de las rutinas de un sistema operativo son imperativos en el diseño y desarrollo de sistemas operativos, sin importar si son tan simples como PintOS (al menos en comparación con otros sistemas, como GNU/Linux, FreeBSD, Solaris, etc.)

El uso de heurísticas, y de consideraciones de los procesos a ejecutar, y de los sistemas en los que éstos son ejecutados, puede ayudar en el diseño de los algoritmos de calendarización, por ejemplo, en un sistema operativo embebido, que está destinado a ejecutar procesos de obtención de datos y envío de datos por red, puede usar un algoritmo calendarizador que desperdicie tiempo del CPU, pero que garantice una pérdidad mínima de transferencia de datos, y ésto puede ser considerado como más deseable en éste sistema que el calendarizador que use una computadora de edición de gráficos, que data su naturaleza es una tarea que hace gran uso del CPU, por ser cálculos matemáticos, de manera que se obtiene una versión más optimizada para sus casos de uso, capacidades de hardrware, etc.

VII. Apéndices

Referencias

[1] Andrew S. Tanenbaum. *Operating Sistems: Design and Implementation*, 3rd ed. New Jersey, USA. Prentice Hall.

[2] Richard Stallman y otros contribuyentes al proyecto GNU. *GNU Coding Standards*, recuperado de la dirección electrónica https://www.gnu.org/prep/standards/standards.html el 22 de abril del 2017

[3] Ben Pfaff, et al. CS140: Operating Systems Course Documentation, Stanford University. recuperado de la dirección electrónica https://web.stanford.edu/class/cs140/projects/pintos/pintos.html el 22 de abril del 2017