Implementación de Algoritmos de Calendarización para el Sistema Operativo PintOS

Kevin Umaña Ortega*, Nicolás Jiménez García**
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
* kev1003@gmail.com, ** nicolas.j2007@gmail.com

I. Introducción

El objetivo de este proyecto es modificar el sistema operativo pintOS (desarrollado por la Universidad de Stanford) para manipular la ejecución de conjuntos de hilos que están enfocados en operaciones de uso intensivo de CPU (CPU bound) o uso de operaciones enfocadas intensivamente en dispositivos de entrada y salida (I/O bound). Esta manipulación de ejecución de hilos se hace por medio de calendarizadores, los cuales fueron desarrollados por los autores y hacen uso de cuatro algoritmos de calendarización. Los cuatro algoritmos de calendarización son: First Come First Served, Shortest Job First, Round Robin y Colas multinivel (MLFQ).

Cada uno de estos calendarizadores tiene enfoques diferentes en cuanto a como tratar y ejecutar los hilos: asignación de CPU y dispositivos de entrada y salida a hilos, tiempo de ejecución, asignación de otros recursos, entre otros. Además cada uno de estos calendarizadores ofrece un rendimiento distinto, donde cada algoritmo puede ser beneficioso para ciertos tipos de hilos. Además cada algoritmo tiene diferente comportamiento con respecto a la prioridad de cada hilo que está en cola de ejecución.

Para probar los calendarizadores se realizaron un conjuntos de hilos. El usuario puede escoger la cantidad de hilos a ejecutar, el porcentaje de hilos de uso de CPU y el porcentaje de hilos de Entrada/Salida de datos o bien si todos son de uso CPU o Entrada/Salida de datos. Finalmente los resultados de la ejecución se almacenan en un archivo, para ser comparados después con ejecuciones de otros algoritmos o a necesidad del usuario.

II. Descripción del problema

Este proyecto presenta el proyecto del control de gran cantidad de procesos que normalmente necesitan ser ejecutados por un sistema operativo, ya que la finalidad principal de un sistema operativo es poder ejecutar procesos a necesidad de los usuarios que utilizan los sistemas operativos para que estos logren hacer tareas con las computadoras.

El control de procesos en un sistema operativo es una tarea importante puesto que existen muchos problemas posibles en la ejecución de procesos como: *deadlocks*(bloque de exclusión mutúa) o *starvation* (muerte de hambre de procesos). Además es de gran importancia porque dependiendo de la implementación realizada se puede llegar a tener una gran eficiencia (bajo tiempo de espera entre procesos) o baja eficiencia (mayor tiempo desperdiciado del CPU). Cada algoritmo de calendarización existente se enfoca en diferentes áreas de optimización y ofrece un mejor rendimiento o beneficios en diferentes áreas.

En este proyecto se aspira a implementar cuatro algoritmos de calendarización: FCFS, SJF, Round Robin y Colas Multinivel. Estos algoritmos se desarrollan para poder compararlos bajo diferentes escenarios (diferente cantidad y tipos de hilos), con el fin de hacer hallazgos importantes en cuanto a eficiencia y aprendizaje de sistemas operativos.

Otros problemas secundarios de este proyecto son: Creación de un sistema para arranque de la computadora (pintOS) desde una unidad de memoria extraíble (USB). Creación de programa para comparación del resultado de hilos para realizar *benchmarking*.

III. Detalles de la implementación

Este proyecto se divide en varias secciones las cuales fueron implementadas de forma diferente. A

continuación se presentan las explicaciones de cada sección desarrollada de este proyecto

III-A. Calendarizadores

Para realizar los cuatro calendarizadores se decidió modificar los archivos: threads.h, threads.c, sync.h, sync.c, interrupt.c y timer.c. En los archivos thread.h/c están definidos e implementados los cuatro calendarizadores en sí. En los archivos syn.h/c están definidas funciones importantes para la sincronización de hilos utilizando semáforos e importantes para los calendarizadores. En el archivo interrupt.c están definidos funciones para el manejo de interrupciones al kernel del sistema operativo. En el archivo timer.c están definidos funciones para controlar el tiempo de ejecución del sistema operativo y ciclos de reloj transcurridos durante la ejecución.

Dentro de threads.c está implementado la función para escoger el tipo de calendarizador deseado entre los posibles. Además están las funciones necesarias para llevar a cabo los diferentes procedimientos que cada uno de los algoritmos requiere como el cálculo de prioridad para el calendarizador de Colas Multinivel (MLFQ) o la predisposición de un hilo cualquiera a liberar los recursos de CPU y entregarlos a otro hilo. Otras funciones incluyen el cálculos por adelantado de tiempo de ejecución de un hilo.

Otros detalles importantes de implementación incluyen: La definición e implementación de creación de hilo para todas las pruebas. El bloqueo y desbloque de hilos. La funcíon de cambio de estado de un hilo (bloqueado, ejecutándose, esperando). El manejo de colas multinivel para asignación de tipos de hilos. Entre otras funciones importantes.

III-B. Conjunto de hilos

Para realizar el conjunto de hilos se creo una prueba dentro del directorio testsíthreads, esta prueba se nombró: prueba-hilo y también se le creó un archivo de código en lenguaje de programación C llamado prueba-hilo.c. En este archivo se definió todo el comportamiento que el conjunto de hilos debe tener.

La implementación incluye la división de creación de hilos por tipo de operaciones a realizar por porcentajes. Además de la creación de todos los hilos solicitados por el usuario y las iteraciones de cada uno. Finalmente almacena los resultados más relevantes en un archivo de bitácora (log). Este archivo contiene las definiciones de los datos asociados a la prueba en sí y a cada hilo por medio de estructuras.

IV. JUSTIFICACIÓN DE DECISIONES DE DISEÑO

Este proyecto necesitó de varias decisiones de diseño, las cuales son presentadas a continuación:

- **Uso de iteraciones en hilos:** Cada hilo creado cuenta en su ejecución cinco iteraciones para repetir su procesamiento. Estas iteraciones realizan el mismo procesamiento cada vez y aplica para tanto operaciones de uso intensivo de CPU como para operaciones de uso intensivo de Entrada/Salida. Esto se planteó de esta manera ya que es algo común entre los sistemas operativos que un hilo tenga varias iteraciones de procesamiento.

- **Uso comando tee para crear *logs*:** En este proyecto, después de cada ejecución de hilos se utiliza el comando de UNIX tee para imprimir el resultado mostrado en pantalla a un archivo definido antes de comenzar la ejecución del sistema operativo. Esto se realizó de esta manera ya que se encontró que PintOS no posee manejo de FILE I/O de C.
- **Uso de interrupciones:** Se desabilitaron las interrupciones para las operaciones que podían causar condiciones de carrera *race conditions*. Para realizar esto se utilizaron candados *locks* por medio del código presente en *sync.h/c*. Esto se realizó de esta manera para evitar problemas de condiciones de carrera y bloqueos de exclusión mutua *(deadlocks)*.
- **Único archivo de calendarización:** Se decidió incluir las cuatro versiones de calendarizadores dentro de un único archivo con el fin de hacer más simple la compilación y el proceso de creación de hilos por parte de los desarrolladores. Este archivo es threads.c dentro de la carpeta /threads.

- **Uso de timer ticks como unidad de tiempo:** Se decidió utilizar la unidad de tiempo *ticks* provista por pintos, la cual equivale a la cantidad de tiempo en segundos multiplicado por la frecuencia a la que opera el sistema operativo. La razón por la cual se decidió utilizar esta unidad es por que ofrece un mejor contexto de operación de la prueba y es adaptable a cambios de frecuencia.

V. COMPARACIÓN ENTRE ALGORITMOS

Esta sección presenta los resultados obtenidos de los cuatro algoritmos sobre el conjunto de hilos para analizar las diferencias y tener un mejor entendimiento de estos algoritmos. A continuaci

VI. Conclusiones

Tras evaluar los resultados obtenidos y finalizar el proyecto, se recalcan ciertos puntos relevantes y recomendaciones para el futuro.

- Asegurar procesos (hilos) por medio de cierres de exclusión mutua es una gran solución para no causar problemas de sincronización u otros problemas de acceso (lectura o escritura) sucia entre hilos. Utilizar este método garantizó el funcionamiento adecuado en las ejecuciones del sistema operativo.
- Las iteraciones de hilos son una buena forma de simular procesos que se ejecutan en sistemas operativos modernos, ya que su orden de ejecución no siempre es lineal y puede ser calendarizado por aparte de la calendarización de hilos. Finalmente cada iteración puede realizar operaciones distintas.
- El valor de un *quantum* debe basarse en estimaciones del tiempo de procesamiento de cada hilo. Y ser superior ligeramente a este de modo que se reduzca el número de veces que se realizar el cambio de contexto para tener un *overhead* pequeño y optimizado.