Programación y Algoritmos I - Tarea 1
Jorge Miguel Valles Silva

Ejercicio 1. Resumen de la siguiente publicación:

S. Kumar and P. K. Singh, "An overview of modern cache memory and performance analysis of replacement policies," 2016 IEEE International Conference on Engineering and Technology (ICETECH), 2016, pp. 210-214, doi: 10.1109/ICETECH.2016.7569243.

Resumen.

El caché es un lugar para almacenar un subconjunto de datos o instrucciones frecuentemente utilizados. Su finalidad es evitar accesar a la memoria principal (una operación costosa) cada vez que la misma información es requerida.

El rendimiento de la memoria caché es calculado mediante la tasa de fallos, la penalización por fallos y el tiempo de acceso promedio. La tasa de fallos se define como la fracción de accesos a la memoria que no se encontraron en el caché. La penalización por fallos se refiere al número de ciclos que el CPU se detiene durante un acceso a la memoria principal. Estas dos variables se utilizan para determinar el tiempo de acceso promedio.

El sistema de memoria en un diseño uniprocesador era un simple componente, hecho de unos pocos niveles de memoria caché (desde uno hasta tres: L1, L2, L3) de datos e instrucciones. En procesadores multi-núcleo cada núcleo del procesador consiste de todos los componentes de un uniprocesador. Los niveles de memoria caché pueden ser individuales de cada núcleo o compartidos. Es por ello que en procesadores multi-núcleo hay tres principales variables: el número de núcleos, niveles de memoria caché y cantidad de memoria caché compartida.

Hay tres unidades principales en un procesador: instrucción, ejecución y unidad de almacenamiento. La unidad de instrucción es responsable de organizar instrucciones de un programa para ser recogidas y decodificadas. La unidad de ejecución se encarga de operaciones lógicas, matemáticas de la ejcución de las instrucciones. La unidad de almacenamiento establece una interfaz a través de un almacenamiento temporal entre las otras dos unidades.

El algoritmo de reemplazamiento juega un papel muy importante en el diseño de la memoria caché porque toma la decisión de seleccionar una línea particular de la memoria caché para ser reemplazada por el bloque de memoria principal deseado. First in First Out (FIFO), Least Frequently Used (LFU), Random y Least Recently Used (LRU) son algoritmos utilizados para tomar dicha decisión. Todas estas políticas, excepto Random, determinan qué bloque de memoria caché reemplazar fijándose únicamente en referencias del pasado.

Para la parte final del artículo, se propuso una metodología de evaluación de los algoritmos de reemplazamiento anteriormente mencionados, usando un simulador de rendimiento, para compararlos basándose en varios puntos de referencia. La concluisión fue que Least Recently Used (LRU) fue el algoritmo más reescalable. También concluye que la velocidad de los procesadores crece continuamente con la disminución de la latencia de memoria.

Ejercicio 2. Describir detalladamente los paradigmas de computación principales y su relación. Dar ejemplos de cada paradigma.

Programación Imperativa: está más orientado al desarrollo de hardware. Los procesadores están diseñados para ejecutar series de ¡¡instrucciones máquina¿¿ consecutivas. Entonces, abstracciones de estas instrucciones han dado paso a lenguajes de mayor nivel.

Ejemplos: lenguajes ensabladores.

Programación Estructurada: constituye una forma simplificada de programación imperativa. La principal modificación del principio básico radica en que, en lugar de instrucciones de salto absolutas (instrucciones que provocan que el procesamiento no continúe con la siguiente instrucción, sino en otro lugar) este paradigma de programación de software prevé el uso de bucles y estructuras de control.

Ejemplos: casi todos los lenguajes.

Programación Procedimental: El paradigma de programación procedimental amplía el enfoque imperativo con la posibilidad de desglosar algoritmos en porciones manejables. Estos se denominan como procedimientos, dependiendo del lenguaje de programación, o también como subprogramas, rutinas o funciones. El sentido y el propósito de esta distribución es hacer que el código de programa sea más claro y evitar las repeticiones innecesarias de código.

Ejemplos: la mayoría de los lenguajes (Fortran, Basic, C, Java, Python).

Programación Orientada a Objetos: un programa de este paradigma es visto como una colección de objetos que se comunican entre sí. Los objetos encapsulan datos y funcionalidades. Son instancias de clases. Este paradigma tiene mayor flexibilidad. El código puede ser reutilizado para cualquier tipo de problema relacionado con el mismo tipo de objetos.

Ejemplos: C++, Smalltalk, Java.

Programación Declarativa: se dclara lo que se debe hacer el programa en vez de cómo lo debería hacer. Permite un mayor entendimiento sobre lo que el código está haciendo.

Programación Funcional: la entidad básica es la *función*. En este paradigma no existe la noción de variable. Las funciones se acercan más a la definición matemática pues el resultado siempre debe ser el mismo para una entrada dada.

Ejemplos: Lisp, Scheme, Haskell.

Programación Lógica: la base de datos de conocimientos se alimenta de afirmaciones y reglas lógicas. Ejecutar un programa es evaluar una afirmación o una búsqueda de valores para los cuales una proposición es verdadera.

Hay una relación entre estos paradigmas, pues la programación imperativa, estructurada y orientada a objetos son derivadas de la programación imperativa. Por su parte, la funcional y lógica son derivadas de la declarativa.

Ejercicio 3. Programa que sume 2 fracciones, la entrada debe ser:

Introducir Fracción 1: a/b

Introducir Fracción 2: c/d

La salida debe ser: a/b + c/d = f/g

donde a-f son números enteros. Nota: Hay que leer sobre entrada de datos con símbolos especiales de la función scanf().

SOLUCIÓN. Se adjunta el código, con algunos comentarios para facilitar la comprensión. En la misma carpeta se encuentra un archivo de texto Readme.txt en donde se especifican las instrucciones de compilación y ejecución.