Trabajo de Arquitectura Avanzada

Integrantes del Grupo:

* Orban Tobias Naim
* Silva Riquelme Axel
* Weigandt Lautaro Nahuel
* Altamirano Alex Lisandro
* Perez Ulises Joaquin

Profesor: Alejandro Burgos

Introducción

En el campo de la informática, la gestión eficiente de la memoria cache es fundamental para optimizar el rendimiento de los sistemas. Este trabajo, realizado por un grupo de estudiantes de la facultad, se centra en los algoritmos de reemplazo de cache, específicamente en los algoritmos LRU (Least Recently Used) y FIFO (First-In-First-Out). Estos algoritmos son esenciales para determinar qué bloques de datos deben ser eliminados para hacer espacio a nuevos datos cuando la caché está llena. A través de la experimentación y el análisis, este estudio busca comparar la eficiencia de estos algoritmos y proponer una hipótesis sobre cuál de ellos proporciona un mejor rendimiento en términos de tasa de aciertos.

Desarrollo

El algoritmo de reemplazo es un elemento crucial en el diseño de la memoria cache. Los dos algoritmos más comunes son:

- LRU Reemplaza el bloque menos utilizado recientemente. Cada línea de la cache incluye un bit de uso que se actualiza cada vez que se referencia una línea. Cuando se necesita espacio, se elimina la línea cuyo bit de uso está en cero, bajo el supuesto de que las posiciones de memoria utilizadas más recientemente tienen más probabilidades de ser referenciadas de nuevo. Esto es fácil de implementar para la asociativa por conjuntos de dos vías. Cuando una línea es referenciada se pone a 1 su bit USO y a 0 el de la otra línea del mismo conjunto. Cuando va a transferirse un bloque al conjunto, se utiliza la línea cuyo bit USO es 0. Ya que estamos suponiendo que son más probables de referenciar las posiciones de memoria utilizadas más recientemente, el LRU debiera dar la mejor tasa de aciertos.

-FIFO: Reemplaza el bloque que ha estado más tiempo en la cache. Este algoritmo puede implementarse fácilmente mediante una técnica cíclica (round-robin) o buffer circular. La técnica round-robin asegura que cada proceso recibe una cantidad limitada de tiempo de CPU en turnos cíclicos, evitando que un proceso monopolice los recursos por mucho tiempo.

Tiempo de CPU

CPU es una abreviación de Unidad Central de Procesamiento. El CPU es un componente básico de la computadora personal u ordenador que procesa datos y realiza cálculos matemáticos-informáticos. El tiempo de CPU es la cantidad de tiempo que el procesador dedica a ejecutar las instrucciones de un programa o proceso. Este tiempo se mide cuando la CPU está activa y trabajando en una tarea específica. Es una medida crucial en la evaluación del rendimiento del software y la eficiencia del hardware.

- Tipos de Tiempo de CPU:

- Tiempo de usuario: El tiempo que la CPU pasa ejecutando las instrucciones del código del usuario (el propio programa).

- Tiempo de sistema : El tiempo que la CPU pasa ejecutando instrucciones del kernel del sistema operativo en nombre del programa (como llamadas al sistema y operaciones de E/S).

- Comparación con Tiempo Real:

- Tiempo real: Es el tiempo total que transcurre desde el inicio hasta la finalización de un proceso, incluyendo todos los retrasos y tiempos de espera. Incluye el tiempo en que el proceso puede estar inactivo, esperando recursos, o siendo preemptado por otros procesos.

- Por ejemplo, si un programa se ejecuta en 10 segundos de tiempo real, pero solo utiliza 2 segundos de tiempo de CPU, significa que la CPU estuvo activa solo durante 2 segundos, y los otros 8 segundos se dedicaron a esperas o actividades no relacionadas directamente con el procesamiento.

- Importancia:

- Evaluar el tiempo de CPU es crucial para optimizar el rendimiento de programas y sistemas.

- Ayuda a identificar cuellos de botella y a entender cómo el software utiliza los recursos del hardware.

- En sistemas multiusuario o multitarea, el tiempo de CPU permite asignar y gestionar mejor los recursos del procesador entre diferentes procesos.

- Medición:

- Herramientas como time en Unix/Linux, el Administrador de Tareas en Windows, y diversos perfiles de rendimiento pueden medir y reportar el tiempo de CPU.

- Las métricas obtenidas pueden ser utilizadas para mejorar el código, balancear la carga en sistemas distribuidos, y realizar ajustes en el hardware o configuraciones del sistema.

Localidad

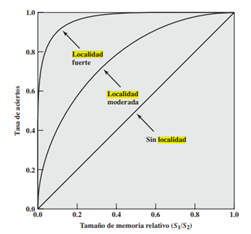
La base para la mejora de prestaciones de una memoria de dos niveles es un principio conocido como localidad de las referencias. Este principio establece que las referencias a memoria tienden a formar agrupaciones (clusters). A lo largo de un período de tiempo largo, las agrupaciones en uso cambian, pero durante periodos cortos el procesador trabaja fundamentalmente con agrupaciones fijas de referencias a memoria.

En la literatura se distingue entre localidad espacial y temporal:

- Localidad espacial: Se refiere a la tendencia durante la ejecución a involucrar múltiples posiciones de memoria que estén agrupadas. La localidad espacial refleja la tendencia del procesador a acceder a las instrucciones secuencialmente y también la tendencia de los programas a acceder a posiciones de datos consecutivas, como por ejemplo cuando se procesa una tabla de datos.

- Localidad temporal: Hace referencia a la tendencia del procesador a acceder a posiciones de memoria que han sido utilizadas recientemente. Por ejemplo, cuando se ejecutan iteraciones de un bucle, el procesador ejecuta repetidamente el mismo conjunto de instrucciones. Tradicionalmente, la localidad temporal se ha explotado manteniendo en memoria caché las instrucciones y los datos usados recientemente, y aprovechando la jerarquía de caché.

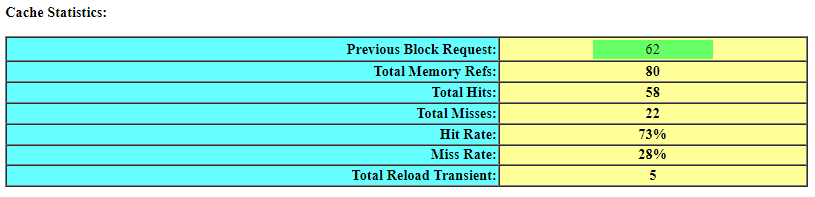
Generalmente, la localidad espacial se explota usando bloques de caché más grandes e incorporando mecanismos de precaptación (captando objetos de uso anticipado) en la lógica de control de caché. Recientemente, se ha investigado bastante para refinar estas técnicas al objeto de conseguir mayores prestaciones, pero las estrategias básicas siguen siendo las mismas. El principio de localidad es fundamental en el diseño de arquitecturas de computadoras y estrategias de memoria, como cachés y la gestión de memoria virtual, ya que permite optimizar el rendimiento al predecir y almacenar datos e instrucciones que probablemente se utilizarán próximamente.

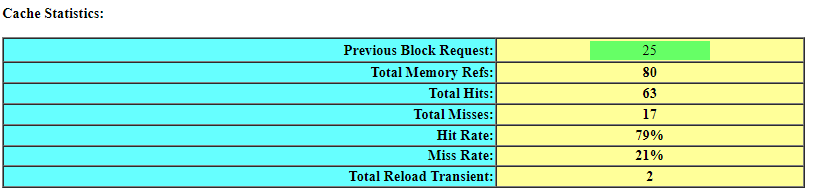


**Pruebas y Resultados**

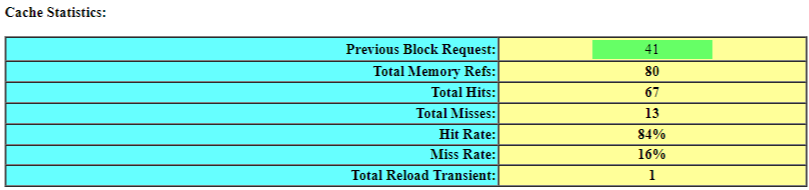
Se realizaron dos pruebas para cada algoritmo, utilizando la misma configuración para asegurar una comparación justa:

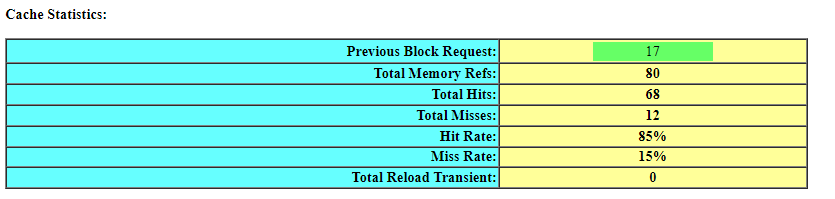
1.Prueba FIFO Nº1 y Nº2





2. Prueba LRU Nº1 y Nº2





Los resultados mostraron que el algoritmo LRU tuvo una mayor cantidad de hits comparado con el FIFO, lo que indica una utilización más eficiente de la cache. Mientras que FIFO tuvo que realizar más recargas de transición debido a la insuficiencia de memoria, LRU solo necesitó una recarga en total, minimizando la ralentización de la ejecución.

Conclusión

Este estudio concluye que el algoritmo LRU proporciona una mayor eficiencia en la gestión de la cache en comparación con FIFO, ya que maximiza la tasa de aciertos y reduce la necesidad de recargas. Nuestra hipótesis es que en entornos donde las referencias a memoria son altamente repetitivas, el LRU debería ser preferido sobre FIFO. Sin embargo, es importante considerar el contexto y los requisitos específicos del sistema al elegir el algoritmo de reemplazo más adecuado.

Hipótesis

La hipótesis planteada es que el algoritmo LRU debería proporcionar un mejor rendimiento que el algoritmo FIFO en términos de tasa de aciertos y eficiencia de la cache, especialmente en entornos con referencias a memoria repetitivas y predecibles.

Conclusiones

En conclusión, la implementación del algoritmo LRU en sistemas de gestión de memoria cache resulta en una mejora significativa del rendimiento en comparación con FIFO. Esto se debe a su capacidad para mantener en la cache los datos más recientes y relevantes, reduciendo así la necesidad de recargas y optimizando el uso de la memoria. La elección del algoritmo de reemplazo debe basarse en el análisis del patrón de acceso a la memoria y las necesidades específicas del sistema. Future investigaciones podrían explorar la combinación de estos algoritmos con técnicas avanzadas de precaptación para mejorar aún más la eficiencia del sistema.