Jun Zhen

CSS 430

11/25/19

**Report**

**Part 1: Specification/design – Cache.java**

In an ordinary operating system, all of the pages of the process will be loaded into the main memory. This is highly inefficient because it big process to hold up the main memory and decrease CPU throughput. A way around this problem is introduce second memory, a backup memory. With Cache.java, we want to simulate the how a page replacement algorithm will impact the CPU throughput by implementing the enhanced second chance algorithm. This algorithm will check each page and look for specific combination of reference bit and dirty bit.

There are four scenarios:

1. Both reference bit is used, and dirty bit been modified. Best choice for replacement
2. Reference bit is not reference but dirty bit been modified. Second best choice for replacement
3. Reference bit is used but dirty bit not been modified. Not the best choice for replacement
4. Reference bit is used, and dirty bit been modified. Do not replace, high chance this page will be used again

The algorithm will loop through the whole memory looking for a page to be removed so that it can load in a new page that is needed for the current process running.

**Functions:**

Cache constructor() – takes in two parameter (blockSize, cacheBlocks) and allocate number of cache blocks and the size of each bytes.

Page() – class/data structor that holds the values of block number, reference bits, and dirtybits. Cache is looking for pages to be swapped out. It will check page’s values to determine if a certain page will be move to secondary or remain.

Read() – takes in two parameter(blockId, buffer). This method will read into the buffer and look for the specified block id from the second memory. If it found the block, then it will return true otherwise false which mean it is a page fault.

Write() – takes in two parameter(blockid, buffer) This method will search for the specific spots in the secondary memory and write data into it.

findVictim() – find a page on the main memory that is most suitable to be swapped out from main memory to secondary memory.

Sync() – maintains the data in main memory and secondary to ensure valid clean copies od data. Used right before the ThreadOS shuts down and writes data to a disk file.

Flush() – clear out all dirty blocks and bad cached blocks. Used during performance test to minimize errors.

**Part 2 Test 4 Results:** A screenshot of a cell phone

Description automatically generated*Test 4 results with cache enabled*

A screenshot of a cell phone

Description automatically generated

*Test 4 results with cache disabled*

**Performance consideration on Random Accesses**

In this test case, we are writing and reading data randomly across the disk. The result from cache enabled and cache disable is very similar. With cached being slightly faster (15279) than non-cached (15642). A reason why the data are so close have to do with system is spending most of the time looking through the scattered data rather than replacing pages.

**Performance consideration on Localized Accesses**

In this test case, we are writing and reading data from small selected section of blocks. With cached enable, it increases the performance significantly. It was almost instance with a score of 1. While cache disable lagging around 8807. Because the data are localized, which mean we are spending less time looking for the data and the transferring from secondary to main is almost instance!

**Performance consideration on Mixed Accesses**

In this test case, we are having 90% of the data being locally accessed and 10% random accessed. From what we learn from localized access, is that it great increases the performance of the system due to fast fetching of data. Although 10% of the blocks are randomly accessed, with cached enabled, it still beats out cached disabled by half. 4401 cached-enabled and 10710 cached-disabled.

**Performance consideration on Adversary Accesses**

In this test case, we are intentionally causing disk access to not make use of caching. The result shows that it in fact slows down the system. The data for both test is very close with cache being disabled finally win a round with 22489 score. While cache enabled did performance worst 23663, it not to disregard the strength of caching. We see in previous tests that when caching is able to work, it works great. In this case, we are implicitly causing the algorithm to miss with unorthodox values for the pages’ reference and dirty bit. This resulted in the algorithm spending great deal amount of time confused, thus greatly decrease its performance.