CSS430 Operating Systems

Professor Stephen Dame

Program4: Cache

Christopher Tran

# Cache Implementation

## Assumptions

-1 BlockID is an empty cache location

-1 return on any function indicates non-existence

## Second Change Algorithm

Implemented in findAvailablePage function

Continuous while loop as there shouldn’t be any situation that would cause infinite execution

Class variable nextIndex is the start of a search for a page and copied to currentIndex scope variable

First checks if the index is empty with a BlockId of -1

increments nextIndex and returns the current index

Checks if the index has a referenced bool of true

sets reference index to false

increments nextIndex and continues to the beginning

if reference is false

Check if dirty and write page to disk using writeToDisk function

incremements nextIndex and returns the current index

## Read Method

Checks if the blockID is less than 0 and returns false as it’s an invalid blockID.

Function checks if the blockID exists in the cache and returns index of location

If the returned index is not -1 then data is copied into the parameter byte array

If the returned value is -1

Function findAvailablePage returns an int of an available page based on index

Returned Int is used to read data from disk using BlockID and stores in location of the returned index with SysLib.rawread

From the cache location, array is copied into the parameter byte array

BlockID is stored in an array to indicate which block is stored

Referenced array based on index is marked true

## 

## 

## Write Method

Checks if the blockID is less than 0 and returns false as it’s an invalid blockID.

Function checks if the blockID exists in the cache and returns index of location

If the returned index is not -1

data is written to the cache location of the index

Referenced based on index is marked true

Dirty based on index is marked true

If the returned index is -1

Function findAvailablePage returns an int of an available page based on index

Using the returned index, copy data from parameter array into the cache location

Set the int for the block stored using parameter BlockID

Set reference to true

set Dirty to true

## Sync and Flush

Private writeToDisk function writes specific indexes in the cache

Both sync and flush loop through the number of pages and uses function to write to disk

Flush does additional tasks

Sets referenced boolean array to all false

sets the blockId stored in each index to -1 to revert to empty

sets Dirty boolean array to all false

Changes the nextIndex to search for when finding a page back to 0

# 

# 

# Performance Results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CACHE OFF : Write to Disk | | | | | |
| Test | Run1 | Run2 | Run3 | Run4 | Run5 |
| Random | 21814 | 22333 | 22279 | 21141 | 21916 |
| Localized | 8152 | 8134 | 8147 | 8085 | 8108 |
| Mixed | 11226 | 12502 | 11547 | 11331 | 11798 |
| Adversary | 15181 | 15192 | 15182 | 15184 | 15185 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CACHE ON: Disk Caching | | | | | |
| Test | Run1 | Run2 | Run3 | Run4 | Run5 |
| Random | 20375 | 20595 | 21940 | 21392 | 21384 |
| Localized | < 1 | < 1 | < 1 | 1 | < 1 |
| Mixed | 5401 | 4590 | 5852 | 4583 | 6462 |
| Adversary | 15029 | 15025 | 15026 | 15024 | 15115 |

Performance Averages

|  |  |  |  |
| --- | --- | --- | --- |
| Averages | cache off | cach on | difference |
| Random | 21897 | 21137 | 759 |
| Localized | 8125 | < 1 | 8124 |
| Mixed | 11681 | 5378 | 6303 |
| Adversary | 15185 | 15044 | 141 |

# 

# 

# Performance Considerations

## Random Access

I wouldn’t expect random access to have any noticeable improvement. The nature of random access would suggest that there would be very little cache hits as the probability would be significantly low.

## Localized Access

Performance difference was expected. Since the only time data is written from the disk to cache is in the beginning when the first read is made, all actions after are done completely within the fast cache. What was surprising is the less than 1 ms average. In fact, they were all showing 0 ms but that would be impossible, the value was just too small to be truncated with a 1.

## Mixed Access

Since the majority of cases should be cache hits, the performance was expected to be improved. However, I expected a 10% random chance to have a small effect but the difference was greater than anticipated. Even with a probability of 10% being random access, the increase from purely localized to 90% localized was a 5 second magnitude.

## Adversary Access

Since the sequence chosen was specifically designed to cause a cache miss on every action, there should be no difference in performance between disk and cache and that was achieved. I expected the difference to be slightly worse on caching since there was additional overhead in checking for the next available page. I can’t explain why the cache ran slightly faster even if it was an insignificant amount.

## Overall thoughts

After analyzing the DISK class, it was clear that random was going to be longer than Adversary due to the higher probability that random numbers will have greater differences in block numbers. The DISK class uses the difference between the current location and the target location to delay action. All of my sequences were mostly 100 blocks apart except every 10th was 900 away. The sum total would be far lower than the sum random differences.

# 

# 

# Test4 implementation

All test begin with creating a byte array of size 512 and populating with random data casted back into byte. since values are absolute, it should be between 0 and 127

## Random

Creates an int array of size 200 to store the random locations in the order they were generated

A for loop that iterates 200 times generates a random number and mods by 1000, the maximum blocks on the DISK

The generated byte data array is then written to the index location

A for loop that reads uses the sequence in the random number array that should now be populated from the iteration of the write for loop.

After reading a block, it checks each index with the original data that was to be written to check data consistency

## Localized

For 20 iterations, data is written and read from the first 10 blocks of the data target.

## Mixed

Similar to random except the random number is checked if the ending number is a 0. If so, a random block within 1000 is selected, for all other ints, a block within 10 is selected.

## Adversary

A sequence was created to populate the 200 integer array.

for(int i = 0; i < 20; i++) {

for(int j = 0; j < 10; j++) {

adversarySequence[(i\*10)+j] = i + (j\*100);

}

}

Array would look like Table 1. Each index read would from left to right and begin again in the next row. Each read forces a track change and in sequence every read is a cache miss.

Table 1: Adversary Array logic

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 |
| 1 | 101 | 201 | 301 | 401 | 501 | 601 | 701 | 801 | 901 |
| 2 | 102 | 202 | 302 | 402 | 502 | 602 | 702 | 802 | 902 |
| 3 | 103 | 203 | 303 | 403 | 503 | 603 | 703 | 803 | 903 |
| 4 | 104 | 204 | 304 | 404 | 504 | 604 | 704 | 804 | 904 |
| 5 | 105 | 205 | 305 | 405 | 505 | 605 | 705 | 805 | 905 |
| 6 | 106 | 206 | 306 | 406 | 506 | 606 | 706 | 806 | 906 |
| 7 | 107 | 207 | 307 | 407 | 507 | 607 | 707 | 807 | 907 |
| 8 | 108 | 208 | 308 | 408 | 508 | 608 | 708 | 808 | 908 |
| 9 | 109 | 209 | 309 | 409 | 509 | 609 | 709 | 809 | 909 |
| 10 | 110 | 210 | 310 | 410 | 510 | 610 | 710 | 810 | 910 |
| 11 | 111 | 211 | 311 | 411 | 511 | 611 | 711 | 811 | 911 |
| 12 | 112 | 212 | 312 | 412 | 512 | 612 | 712 | 812 | 912 |
| 13 | 113 | 213 | 313 | 413 | 513 | 613 | 713 | 813 | 913 |
| 14 | 114 | 214 | 314 | 414 | 514 | 614 | 714 | 814 | 914 |
| 15 | 115 | 215 | 315 | 415 | 515 | 615 | 715 | 815 | 915 |
| 16 | 116 | 216 | 316 | 416 | 516 | 616 | 716 | 816 | 916 |
| 17 | 117 | 217 | 317 | 417 | 517 | 617 | 717 | 817 | 917 |
| 18 | 118 | 218 | 318 | 418 | 518 | 618 | 718 | 818 | 918 |
| 19 | 119 | 219 | 319 | 419 | 519 | 619 | 719 | 819 | 919 |