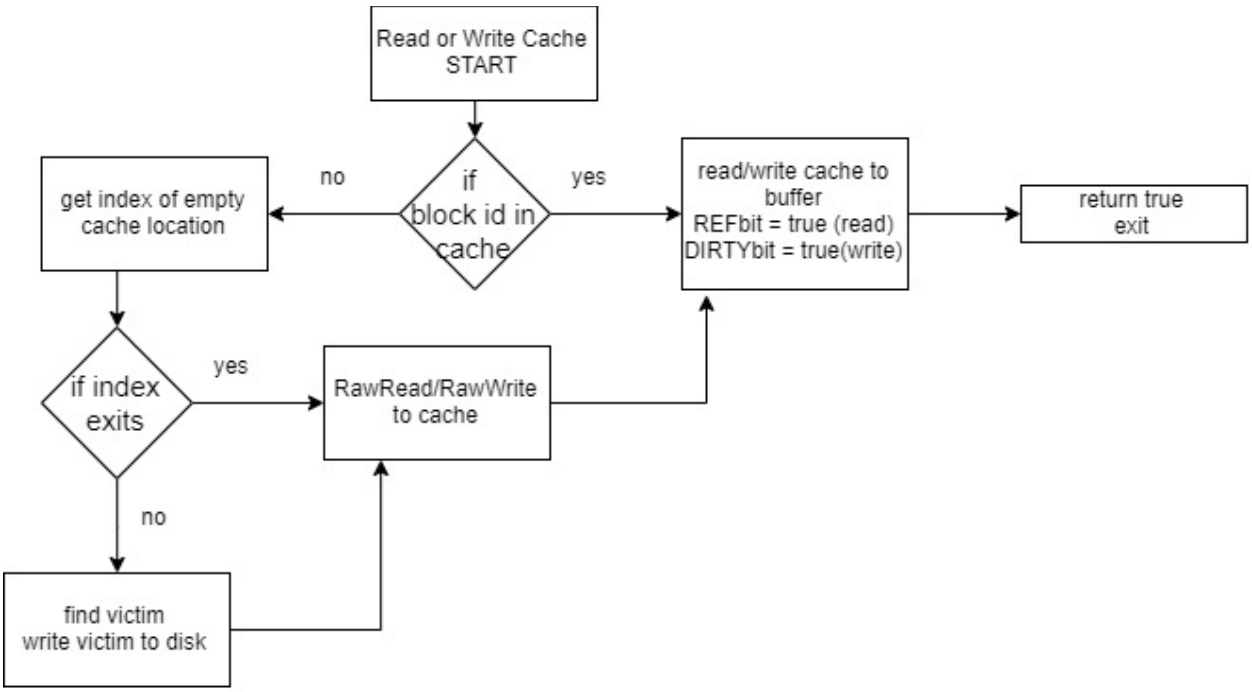
Sharanya Sudhakar

11/24/19

Disk Caching Report

Cache.java



Assumptions and Properties:

A private cacheblock class is created. This cacheblock class is the underlying data type for the Cache

CacheBlock:
Byte[] data
Int framenumber
Boolean reference bit
Boolean dirty bit

The Cache is then initialized with the cache block size, the cache size and the initial victim.

READ /WRITE CACHE

The steps in read/ write to cache is the same, the read or write operation is interchangeable.

Given the blockid, check my cache to see if there is a cache hit. If yes, the read/write operation is performed. If read, ref bit is set to true, if written, dirty bit and ref bit are set to true.

If the cache hit fails, the next available empty cache location is found. If this location exits, the system read/writes to cache, which is in-turn read or written to buffer. If an empty location does not exit, the victim is found based on the enhanced second chance algorithm. The victim is written to disk, if it has a dirty bit and then is used by the new read/write command. First a rawread/rawwrite action followed by a read/write to buffer. The bit are updated as before, in-case of a read, ref bit is set to true, if written, dirty bit and ref bit are set to true.

FIND VICTIM

The victim is set at -1. To begin with… once the function is invoked, the victim is incremented to the next index in the array. This is the pointer to the array. In accordance, the function exits at the last index always, so that every time it is invoked the function will increment the victim index.

There are two variables, count set to 0 will keep track of the loop index, and is00 will keep track of which condition to victimize. This is alternated between 00 and 01. If 00 cannot be found then a 01 condition is found as simultaneous ref bits are cleared. If no 01 victims are found and the ref bits are cleared, then the loop switches back to look for 00. Therefore, the loop switches back and for the looking for 00 and 01 alternately. When there is no cache hit, the victim page does not advance. The victim page only advances when there is a cache hit and a victim is selected. When the victim is selected, page after the victim page now becomes the new pointer value.

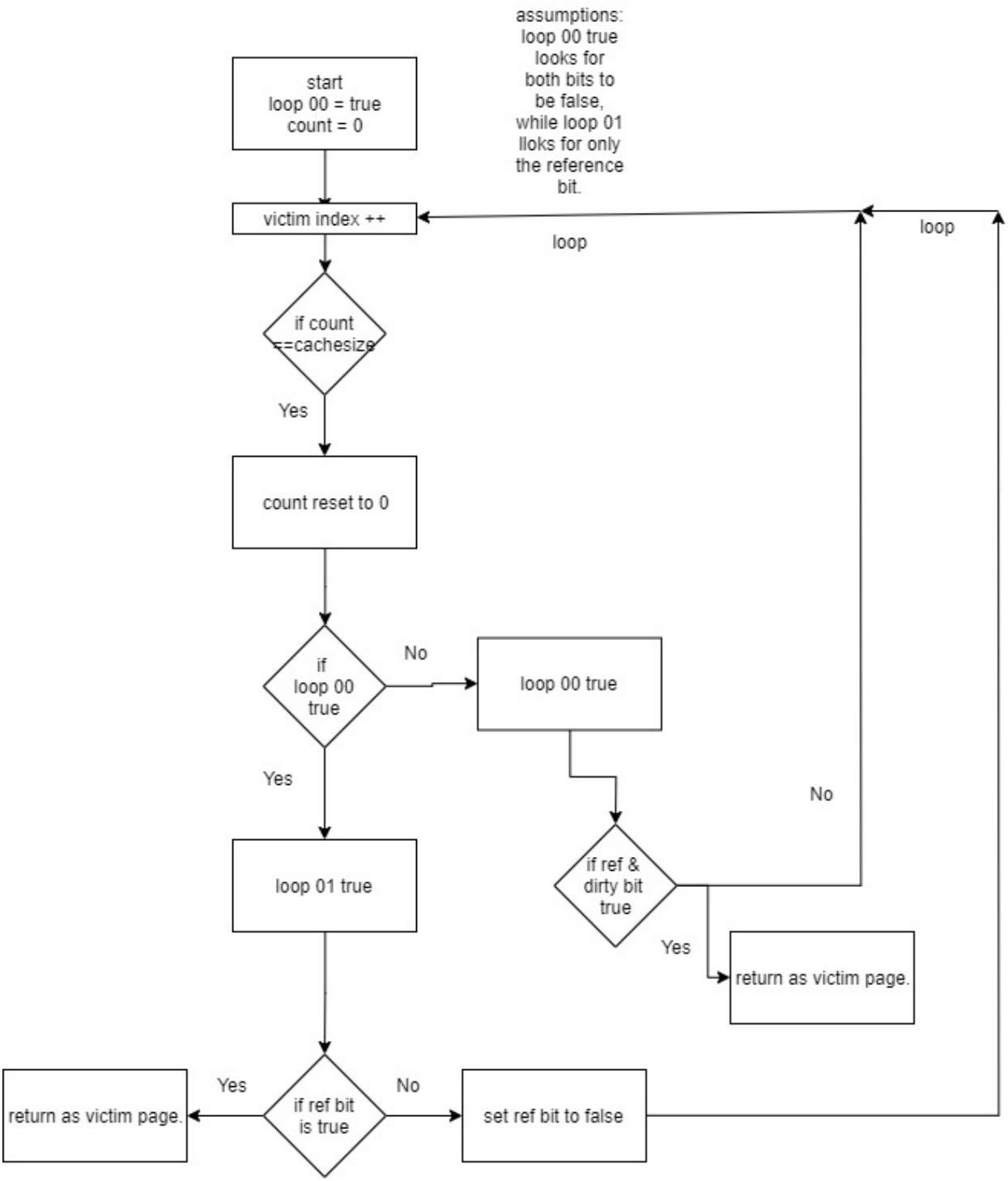
Note: the ref bits are cleared only on the second loop.

FLUSH

The cache bits are reset to empty block number and false for both reference and dirty bits.

SYNC

The locations with dirty bits are written to disk and cleared of their dirty bits. This function is used before exiting the test.


assumptions:
loop 00 true
looks for
both bits to
be false,
while loop 01
lloks for only
the reference
bit.
start
loop 00 = true
count = 0
victim index ++
loop
loop
if count
==cachesize
Yes
count reset to 0
if
loop 00
true
No
loop 00 true
Yes
No
loop 01 true
if ref &
dirty bit
true
Yes
return as victim page.
return as victim page.
Yes
if ref bit
is true
No
set ref bit to false

TEST4

This class tests the cache using different variations of the read and write. It takes 2 arguments. Enabled/disabled and 1/2/3/4/0.

Enabled indicates using the cache. Disabled o the other hand is a direct system read, write.

1 Random Access: This method randomly accesses the cache to read and write to it. The block id is a random number from 0-5000. And the number times a read/write function is executed is between 100-200

2 Localized Access: This method only access within the cache. All access are random with block id from 0-9 and the number of access is a random number between 100-200.

3 Mixed Access: a random number between 100 and 350 is split in a 90:10 ratio. With 90 localized access and 10 random access.

4. Adversary Access: Here the block id is a random number in 1000 so that there is almost never a cache hit. The number of access counts are between 100-300.

0 ALL ACCESS: test case where all the access are performed with a flush command after each. So a new cache is utilized for every access group.

Discussion:

With the given results it is easy to conclude that with the cache the performance is better.

For random access: The difference between enabled and disabled in not by much just 20 or so msec. This is mainly because the cost of randomly accessing from the cache is very similar to random disk access.

For localized access: the cache has almost no time while the disk cache is slightly more expensive.

For mixed access: it is implicit that having cache is more performance enhancing than random calls.

For adversary access: the cache hits are non existent and continuous blocks of memory are not accessed. The cache is useless in this case, as there are seldom any cache hits. This is also most similar to reading and writing on disk, even then the disabled mode is more expensive than a cache mode.

Having more cache hits and a good size cache is always better for performance than random access or adversary access. In either case they are the worst case scenarios and most of the time have no workarounds.

```
-->l Test4 disabled 0
l Test4 disabled 0
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=0)

Random Access...130 With Caching Disabled
WriteTime: 13748 Readtime: 7957
Avg. WriteTime: 105 Avg. Readtime: 61

Localized Access...202 With Caching Disabled
WriteTime: 4302 Readtime: 4093
Avg. WriteTime: 21 Avg. Readtime: 20

Mixed Access
Localized Access: 90%...270 With Caching Disabled
WriteTime: 14635 Readtime: 10562
Avg. WriteTime: 54 Avg. Readtime: 39

Random Access:10%...31 With Caching Disabled
WriteTime: 4319 Readtime: 2857
Avg. WriteTime: 139 Avg. Readtime: 92


Adversary Access...275 With Caching Disabled
WriteTime: 42163 Readtime: 20988
Avg. WriteTime: 153 Avg. Readtime: 76
```

```
sharanu@uw1-320-01:~/U/ThreadOS$ java Boot
threadOS ver 1.0:
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test4 enabled 0
l Test4 enabled 0
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)

Random Access...265 With Caching Enabled
WriteTime: 18186 Readtime: 10177
Avg. WriteTime: 68 Avg. Readtime: 38

Localized Access...164 With Caching Enabled
WriteTime: 236 Readtime: 0
Avg. WriteTime: 1 Avg. Readtime: 0

Mixed Access
Localized Access: 90%...306 With Caching Enabled
WriteTime: 10738 Readtime: 5581
Avg. WriteTime: 35 Avg. Readtime: 18

Random Access:10%...35 With Caching Enabled
WriteTime: 3350 Readtime: 2403
Avg. WriteTime: 95 Avg. Readtime: 68


Adversary Access...102 With Caching Enabled
WriteTime: 10624 Readtime: 6012
Avg. WriteTime: 104 Avg. Readtime: 58
```