Java Minibase Report

Team AQ - Mokhles Bouzaien and Laila Salhi

25th November 2019

1 FIFO and LIFO

FIFO and LIFO implementations are very similar to LRU and MRU ones. So for example to implement FIFO, we added the FIFO class to the BufMgr package and the added the constructor and the class methods:

- *void setBufferManager(BufMgr mgr)*: this method uses the same *setBufferManager* of the *Replacer* class to select the buffer manager, get the total number of frames and set their state to *Available*. In addition, it initialize *frames* and *nframes*: two data structures used in *FIFO* to keep information about pages available in the buffer pool.
- *void pin(int frameNo)*: used to pin a page. Already implemented in *Replacer* class.
- *void update(int frameNo)*: this method is uded to update the *frames* list: it searches for the referenced page and move it to the very end of the list.
- *int pick_victim()*: when no more frames are available in the buffer pool, this method is used to select a victim page and replace it with the newly referenced page: it searches for the first unpinned page in *frames*, replaces it and calls *update* to move it to the end of the list.
- *String name()*: simply returns the name of the replacement policy.

The *LIFO* class implements exactly the same methods. The only difference is that *LIFO* replacement policy places the newly referenced pages at the beginning of *frames* list in order to be picked first.

To implement *FIFO* and *LIFO* we took the code from MRU and LRU and just removed the *update* part in method *pin*.

2 LRU-K

Why LRU-K?
The idea of the *LRU* replacement policy is that the most recently used pages are most likely to be used during the next instructions. So when replacing, the least recently used pages are chosen as victims. The pages are stored in an ordered list called *frames* and containing *n=nframes* pages (the same structures used previously): the least recently used element is at the back of the list. This replacement policy has a few drawbacks such as:

- It does not give importance to usage frequency of page, so it can keep pages that have very infrequent reference.
- We have to update the list about every memory reference which is time consuming.

The *LRU-K* replacement policy is an extension of the *LRU* that uses the last K references to a page ($K \geq 2$) to pick a victim and only keep pages with shortest access interval time or in other terms a greatest probability of reference. Do do that, we need to calculate Backward K-distance $b_t(p, K)$ which is the time T such as $r_T = p$ and p is referenced exactly $K - 1$ times after T. If there are not K references of p in total, then $b_t(p, K) = \infty$.
In this policy, two structures are used to keep information about pages: $Hist(p, k)$ and $Last(p)$. Those two structures will be discussed later[1].

2.1 $Hist[p][k]$

In the case where a page is not in the buffer pool: that means that no *frameNo* was given to the page and that means that the *frameNo* (*pageId* in the buffer pool) is not in *frames* the list of *frames* used from the bufferpool. In *LRU* policy, the *frameNo* is added and shifted to the end of *frames*. In the particular case of *LRU-K* policy, we also add a line to the matrix *Hist*. It's matrix (multidimensional array in *Java*) of size $numBuffers \times K$ containing at a given position $(p, k) \in [0, numBuffers - 1] \times [0, K - 1]$ the timestamp (in milliseconds) of the $(k + 1)^{th}$ occurrence of page p backwards (e.g. $Hist(p1, 0)$ is the last reference timestamp of page $p1$).

2.2 $Last[p]$

last is the second data structure used to get track of timestamps of the k^{th} occurrences of p without taking into account the correlation condition. That means whenever the page p is referenced, *last[p]* is updated with the reference time.

3 Changes to other classes

Minor changes were performed to other classes in order to adapt our code to the new implementations.
BufMgr: In the *BufMgr* class, we added *private int lastRef* which representes the K parameter of *LRUK* replacement policy. We also changed the signature of the *BufMgr* class by adding the *int lastRef* parameter and added a *lastRef* getter in order to use this parameter in *LRUK* class.
BMTest2: a constant value *public static final int K = 2* was added to perform tests with. Now *SystemDefs* requires the K in order to create a *BufMgr* instance (using the *init* method). The *replacealgo* list was updated in order to test different replacement policy algorithms.

References

[1] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. *The LRU-K page replacement algorithm for database disk buffering.* SIGMOD, 1993.