EURECOM

DBSYS PROJECT

Notes

DataBase System Management

Student Authors
Irina MOSCHINI
Thiziri NAIT SAADA

Teacher
Paulo PAPOTTI

1 What we did

1.1 FIFO

We removed the systematic update while pinning a page, that was used in the LRU algorithm. We obvisouly kept the track of the order of the frames entering the buffer pool thanks to the frames list. The *pickvictim* function remains unchanged.

1.2 LIFO

We just inverted the order of the LRU algorithm in which we check if there is any unpinned page available. This was done by starting the checking loop to index $nframes - 1$ and decreasing the index until the first one. This way, this is the last-in unpinned frame that is picked as a victim.

1.3 LRU-K

We changed the *pickvictim* function so that the victim frame picked is : - the first one available if the buffer pool is not full - if the buffer pool is full, the first unpinned frame that has not been called at least k times - if the buffer pool is full and not any frame satisfies the last condition, the frame with the least k-occurrences

It is important to check these conditions in this order. Then, we implemented an *update* function that updates the hist and last lists, according to the pseudo code.

2 What we learned

2.1 Difference between FIFO and LRU

Implementing FIFO enabled us to understand the meaning of "pinned" and "unpinned" pages. We realized especially that a pinned page is a page that is being used by the CPU and sometimes, even if it is not a pinned page, it can be kept in the buffer pool. This being said, the main and only difference between these two algorithms is how they deal with updating the frames list when a page already in the buffer pool is pinned. Hence, FIFO consisted only in removing the systematic update function call in the pin function, so that the update function is called only when a new page not in the buffer pool is pinned.

2.2 LRU-k pseudo code

The article was really helpful to compute the LRU-k algorithm. Morevoer, simplified LRU-k with a correlated reference period equal to zero enabled us to get a first simple implementation of LRU-k. Once we achieved this and dealed with all the changes needed in the *pickvictim* function, related to this new class, we added the non zero correlated reference period to our code. Dividing the implementation into these two parts was really meaningful.

2.3 Adapting the whole code with LRU-k's implementation

We changed the buffer manager constructor by adding an attribute : int lastRef. This being done, we needed to change the SystemDefs and BMTest2 functions to take into account this change. When LRU-1 is called, the replacer chosen is simply LRU. Otherwise, LRU-k replacer is called, with k equals to $lastRef$.

We overcame some errors by realizing that the lastRef value was always 0 (thanks to the use of many prints to understand what was going on) because of a careless mistake.

2.4 The database initialization

Trying to understand how the database was initialized during the test4 of BMTest2, we realized that the database was initialized in a quite strange way (init function in DB.java). Especially we noticed that after the database is created in the SystemDefs class, it is initialized in the DB class, where a "first page" is pinned and then unpinned. However, this very page is never freed or deallocated, hence our buffer pool has kept it, and our $nFrames$ starts at 2 for the tests.

2.5 Assumption of dealing with pages

We didn't see this assumption and tried at first to implement the "right" version of LRU-K, dealing with pages. It was finally too complicated so we used only data structures referring to frames, assuming it plays the role of pages in the pseudo-code.