EURECOM

Martin Guyard - martin.guyard@eurecom.fr

Guillaume Brizolier - guillaume.brizolier@eurecom.fr

DBSys Assignment Notes

For Monday 25th November

FIFO Replacement Policy

For the FIFO replacement policy, our approach was quite simple: a FIFO is basically an LRU that doesn't get updated. The 'frames' data structure keeps track of the order of entrance of the frames in the buffer pool. When it is full, the oldest frame (the one at the start of the queue) gets picked for eviction, and all other frames advance of 1 in the queue. In the case where all potential victims are pinned, we throw an exception.

LIFO Replacement Policy

For the LIFO policy, like the FIFO we do not update the frames data structure when pinning, but only when we pick a victim to evict from the buffer pool. When it is full, instead of searching from the start of **frames[]**, we start looking from the end so as to implement a stack. This ensures that the latest frame to have entered the buffer pool gets evicted. In the case where all potential victims are pinned, we throw an exception.

LRU-K Replacement Policy

For the LRU-k, simply keeping an array as the only data structure to hold the frames won't do. We decided to use:

- A **Hashmap<Integer, ArrayList<Long>>** to keep track of the access history of each frame
- A **Hashmap<Integer, Long>** to store the latest accessed time of each frame.

In the LRU-K policy, we update the history of the frame when it is pinned, depending on the correlated reference period, by pushing the current time into the history ArrayList.

In the **pick_victim** function, we then proceed to find the Kth value in the history of the given frame. The victim is then picked as the frame that has the biggest Kth value in its history. In the case where all potential victims are pinned, we throw an exception.