**Design**

Mi Tian, Yuchen Liu

Task 2.1:

There are two methods for efficient finding of Empty Frame: keep a free list or make a bit map, in this project, we choose the first one. *freelist* in this project is a stack of integer (buffer ID). When initialize a new BasicBufferMgr, we create a new *freelist* with all the buffer id push in, since in the beginning all buffers are free to use. In function *Pin*, if a buffer is used, get this buffer id and remove from the *freelist.* When system need to choose an empty buffer to use, pop one from *freelist* if *freelist* is not null. In order to printing out the information about this *freelist* for testing purpose*,* we add a helper function *getFreelist* to return *freelist.*

Task2.2:

In order to search for a given disk block efficiently, we implement a hash table *bufferPagesinPool*, which take an integer as key (block’s hash code) and also a integer as value (buffer id). When initialize a new BasicBufferMgr, we create a new *bufferPagesinPool* with nothing in it, since at first no block has been put in to buffer. If pin a block has just been assigned to a buffer, get this block’s hash code as the key and buffer id as the value, put into *bufferPagesinPool*. When we want to find whether a given block exists in the buffer, hash table is certainly an efficient tool for this, we can check whether this hash table contains this key, if the answer is yes, return the value which is the frame this block was assigned to.

Task 2.3

For the replacement policy, the code is in the *replacementPolicy* package, there is a interface about replacement policy, since both LRU and Clock need to implement *chooseBufferForReplacement* function. *chooseBufferForReplacement* should be used when we want to choose a buffer to place a block and free list is empty at that point.

For LRU, in the class *buffer*, we keep track of last modified time of this buffer by using *lastModified* which is a long value and is initialized to 0 at first, in *LRUPolicy*, we store last modified time of first buffer in variable *lruDate*, and loop each buffer in the buffer pool, compare each buffer’s last modified time and store the oldest one in *lruDate*, after the loop, the buffer with the oldest modified time can be found and the function should return its buffer id.

For Clock policy, in the class *buffer*, we keep track of reference bit of each buffer by using *referenceBit* which is an int value and is initialized to 0 at first, we also add two functions: *setReferenceBit*, *unSetReferenceBit* to set ref-bit as 1 or 0 and *isReferenceBitset* to check if the current buffer’s reference bit is 1. In ClockPolicy, we keep track of current clock pointer’s position by implementing *clockPointer* and initialize to 0 at first. Loop each buffer in the buffer pool. If buffer is pinned, skip this buffer and check next buffer. For the unpinned buffers, if this buffer’s reference bit is 1, set it 0 and skip, until first unpinned buffer with reference bit is 0 is found, return this buffer id.