**Team Members**

Preyang Shah (Leader) - A20385823 - pshah115@hawk.iit.edu

Riten Chhatrala - A20385102 - rchhatrala@hawk.iit.edu

Meet Patel - A20379333 - mpatel104@hawk.iit.edu

Niket Patel - A20384264 - npatel102@hawk.iit.edu

**File List**

Makefile

buffer\_mgr.c

buffer\_mgr.h

buffer\_mgr\_stat.c

buffer\_mgr\_stat.h

dberror.c

dberror.h

dt.h

storage\_mgr.c

storage\_mgr.h

test\_assign2\_1.c

test\_assign2\_2.c

test\_helper.h

**Problem Statement**

Aim of the assignment is to implement buffer manager. Fixed number of pages in memory are managed that represent pages from a page file which is managed by the storage. Buffer manager manages memory pages called “Page Frames”.

For each file, there can only be one buffer pool. Buffer manager should be able to handle more than one open buffer pool at the same time. When the buffer pool is initialized, each buffer pool uses one page replacement strategy that is determined.

We implement FIFO, LRU, LRU\_K, Clock and LFU replacement strategies.

**Data Structures**

In Buffer Pool, few attributes are defined. We have implemented a structure named BMInfo, and assigned it to the **BM\_BufferPool**->mgmtData. Attributes of the structure are as follows:

1. pageToFrame - An array from page Number to frameNumber is indicated

2. frameToPage - An array from frameNumber to page Number is indicated

3. numFrames - Filled number of frames in the frame list is indicated

4. numRead- Total number of read done on the buffer pool is indicated

5. numWrite- Total number of write done on the buffer pool is indicated

6. countPinning- Total number of pinning done for the buffer pool is indicated

7. startData- Value of BM\_BufferPool->startData is indicated

8. dirtyFlags - An array of dirtyflags of all the frames is indicated

9. fixedCounts - An array of fixed count of all the frames is indicated

10. Pagehistory - History of reference of each page in memory is indicated

We implemented the frames as a double-linked list. Each node of the frames is called **FrameNode**, and it contains below attributes:

1. pageNum - The page number of the page in the page File.

2. frameNum - The number of the frame in the frame list.

3. dirty - The dirty bit of the frame.( 1 = dirty, 0 = not dirty).

4. fixCount - fixCount of the page based on the pinning/un-pinning request.

5. rf - Reference bit per node to be used by any page replacement algorithm, if required.

6. data - Actual data pointed by the frameNode.

**BufferPool Functions**

**initBufferPool:**

1. BM\_BufferPool instance is taken as an argument by this function and initializes its attributes based on other arguments.

2. This function, first of all, checks the arguments provided. If invalid, returns an error message. Then, attributes of BM\_BufferPool instance is initialized.

3. Frame list with empty frames are initialized thereafter.

**shutdownBufferPool**:

1. BM\_BufferPool instance is taken as an argument by this function and deallocates all the memory.

2. This function, first of all, checks the arguments provided. If invalid, returns an error message. Then, frames of frame list are emptied and memoery of each node is deallocated.

3. Memory of BM\_BufferPool->mgmtData is deallocated, which refers to the bmInfo structure

**forceFlushPool**:

1. BM\_BufferPool instance is taken as an argument by this function and writes all the dirty frames to the file on the disk.

2. This function, first of all, checks the arguments provided. If invalid, returns an error message.

3. Frame list starting from head is iterated. If any frame has dirty bit and set as 1 :

a. Data is written back to the file on disk

b. Dirty bit is set as 0 and value of numWrite is increased by 1.

4. Returns RC\_OK once all the frames are iterated and if no error is found.

**Page Management Functions :**

**Pin Page**:

1. The main aim of this function is to call pinpage functions which are defined by different strategies such as: pinpage\_FIFO, pinpage\_LRU, pinpage\_CLOCK etc.

2. Also, it pins the page with a particular given page number.

3. Different strategies are used by the Buffer Manager to locate the page requested and also to provide the details of the page to client.

**Unpin Page**:

1. The main aim of this function is to iterate through the available pages in the frames to locate the page which is to be unpinned.

2. The function returns an exception “RC\_NON\_EXISTING \_PAGE\_IN\_FRAME”, if the page is not found.

3. This function returns RC\_OK if the page is found and also “fix Count” is decreased by 1.

**MarkDirty**:

1. The main aim of this function is to iterate through the available pages in the frames to locate the pages which are to be marked as dirty.

2. This function returns RC\_OK if the page is found and also sets the dirty bit of the page node as 1.

**forcePage**:

1. The main aim of this function is to iterate through the available pages in the frames to locate the page which is to be forced to disk.

2. This function opens the file and writes the current content of the page back to the page file on disk, if the page is found.

3. This function returns RC\_OK, if the page is found and the write operation is successful. It returns RC\_NON\_EXISTING\_PAGE\_IN\_FRAME , if the page is not found.

**Statistics Functions:**

This functions provide information regarding the buffer pool and also its contents. Statistic functions are used by print debug functions to provide information about the pool

**getFrameContents**:

BM\_BufferPool->mgmtData includes frameToPage . It is an array of PageNumbers where the ith element was the page stored in the ith page frame. Whenever a new frame is added in the function, this is updated in the function updateNextNewFrame.

**getDirtyFlags**:

1. The main aim of this function is to return an array of Booleans where the ith element is true , if the ith page frame is dirty.

2. BM\_bufferPool->mgmtData->dirtyFlags stores the array. Array is returned by getDirtyFlags.

3. dirtyFlags is populated by traversing the list of frames and checking to see which frames are marked as dirty.

**getFixCounts**:

1. The main aim of this function is to return an array of integers where the ith element is the fix count of the page stored in the ith page frame.

2. For empty page frames, 0 is returned.

3. BM\_bufferPool->mgmtData->fixedCounts stores this array. Array is returned by getFixCounts.

4. fixedCounts is populated by traversing the list of frames and using the fixCount value of each frame.

**getNumReadIO**:

1. The main aim of this function is to return number of pages that have been read from disk since a buffer pool was initialized.

2. Also, this function returns value of BM\_bufferPool->mgmtData->numRead which is set in updateNextNewFrame and in initBufferPool.

**getNumWriteIO**:

1. The main aim of this function is to return the number of pages that have been written to the page file since the buffer pool was initialized.

2. Also, this function returns value of BM\_bufferPool->mgmtData->numWrite which is set in updateNextNewFrame, initBufferPool, forcePage, and forceFlushPool.

**The Page Replacement Strategies:**

**pinPageUsingFIFOStrategy**:

FIFO page replacement strategy is implemented by this function.

1. This function, first of all, checks to see if the page is in memory. Function returns RC\_OK if the page is found and also it calls the searchPageInMemory function described in “Helper Functions” later.

2. First free frame is located starting from head, if the page is required to be loaded in the memory. Page is loaded in the lookUp node and page details are set, if an empty frame is found. Also, the lookUp node is updated to be the head of the linked list.

3. The function starts iterating from tail to the list to locate the oldest frame with fix count 0 for the new page, if all the frames are filled. The lookUp node is updated to be the head of the linked list.

4. If the frame is found following above strategy then updateNextNewFrame function described in "Helper Functions" later; otherwise the function returns no more space in buffer error.

**pinPageUsingLRUStrategy**:

LRU replacement policy is implemented by this function.

1. This function, first of all, checks to see if the page is in memory. It calls searchPageInMemory function described in Helper Functions later and returns immediately with RC\_OK, if the page is in memory.

2. Frame is moved to the head of the framelist everytime it is referenced. The head will be the latest used frame and the tail will be the least used frame at any moment of time.

3. It starts iterating from the tail of the list to look for a frame with fixcount 0, if the page is not in memory.

4. It calls the updateNextNewFrame function described in Helper Function later, if any such frame is found. Otherwise, it returns no more space in buffer error.

**pinPageUsingLRU\_KStrategy: (Additional Feature)**

LRU\_K replacement policy is implemented by this function.

1. This function, first of all, checks to see if the page is in memory. It calls searchPageInMemory function described in Helper Functions later and returns immediately with RC\_OK, if the page is in memory.

2. The reference number (current count of pinning) is updated in the history array (bminfo->pageHistory) everytime a frame is referenced.

3. It starts iterating from the head of the list and calculates the distance as the difference of current count of pinning and kth reference of the page for all pages in memory having fix count 0, if the page is not in memory.

4. The page with the max distance is replaced. It works same as LRU and checks for the least recently used page, if no page is called k time. (kth reference is -1 for all pages).

5. It calls the updateNextNewFrame function described in Helper Functions Later, if any such frame is found. Otherwise, it returns no more space in the buffer error.

**pinPageUsingCLOCKStrategy: (Additional Feature)**

Clock Replacement Policy is implemeneted by this function.

1. This function, first of all, checks to see if the page is in memory. It returns immediately RC\_OK, if the page is in memory.

2. It looks for the first frame with a reference bit that is equal to zero, if the page is not in memory. All the reference bits are set to zero. Reference bit (rf) is set in frameNode->rf .

3. lookUp Node’s new value used in updateNextNewFrame.

**pinPageUsingLFUStrategy: : (Additional Feature)**

LFU Replacement Policy is implemented by this function.

1. This function, first of all, checks to see if the page is in memory. It calls the searchPageInMemory Function described in “Helper Function” later, if the page is found and returns RC\_OK.

2. First free frame is located starting from head, if the page is required to be loaded in memory. Page is loaded in the lookUp node and page details are set, if an empty frame is found. lookUp node is updated to be the head of the linked list.

3. Function starts iterating from trail of the list to locate the frame having the page with minimum page frequency, if all the frames are filled for new page. lookUp node is updated to be the head of the linked list.

4. If the frame is found following above strategy, then updateNextNewFrame function described in “Helper Function” later; otherwise the function returns no more space in buffer error.

**Test Case:**

File Name: test\_assign2\_1.c

This test file will test LRU and FIFO replacement strategies.

**Additional Test Case:**

File Name: test\_assign2\_2.c

This test file will test LRU\_K, CLOCK and LFU replacement strategies.

It also contains additional error checks as mentioned below.

**Additional Error checks :**

Below are the error cases that are checked and tested :

1. Initializing an invalid buffer pool will lead to error. (If no. of frames are negative or 0 or invalid strategies)

2. Pinned a page into a full buffer pool.

3. Pinned an invalid page (uninitialized page instance, or a negative page number.)

4. Unpinned a page which is not available in framelist.

5. Forceflushed a page which is not available in framelist.

6. Markdirty a page which is not available in framelist.

7. Unpinned a page which is available in the frame list, but not pinned by any one.

8. Shutdown a buffer pool which is not initialized.

**Running Information**

1. **make clean**

2. **make run1**

will run test case file test\_assign2\_1.c

3. **make run2**

will run test case file test\_assign2\_2.c