Lab 2: Buffer Pool 4

Instructor: Beom Heyn Kim

beomheynkim@hanyang.ac.kr

Department of Computer Science

Outline

- Buffer Pool Manager
- Assignment

Buffer Pool Manager

buffer_pool_manager.cpp :: FetchPageImpl()

```
Page *BufferPoolManager::FetchPageImpl(page_id_t page_id) {
  // 1.     Search the page table for the requested page (P).
  // 1.1    If P exists, pin it and return it immediately.
  // 1.2    If P does not exist, find a replacement page (R) from either the free list or the replacer.
  //        Note that pages are always found from the free list first.
  // 2.     If R is dirty, write it back to the disk.
  // 3.     Delete R from the page table and insert P.
  // 4.     Update P's metadata, read in the page content from disk, and then return a pointer to P.

  latch_.lock();

  std::unordered_map<page_id_t, frame_id_t>::iterator search = page_table_.find(page_id);
  Page* retPage;
  if (search != page_table_.end()) {
    frame_id_t fid = search->second;
    retPage = &pages_[fid];
    retPage->pin_count_++;
    replacer_->Pin(fid);
    latch_.unlock();
    return retPage;
  }
```

Protecting critical sections

page_table_ contains the mapping between a page id and a frame id. It is used to check if the requested page is already buffered

This part of the code finds the requested page if buffered already, increases the pin count, pins the page and finally returns the pointer to the page

Buffer Pool Manager (Cont.)

buffer_pool_manager.cpp :: FetchPageImpl()

```
// The page requested is not in the memory yet.
// Find the frame to contain the requested page.
frame_id_t pageFrameID;
Page *replPage;
if (!free_list_.empty()) {
  pageFrameID = free_list_.front();
  free_list_.pop_front();
  page_table_.insert({page_id, pageFrameID});
  replPage = &pages_[pageFrameID];
  replPage->page_id_ = page_id;
  replPage->pin_count_ = 1;
  replPage->is_dirty_ = false;
  replacer_->Pin(pageFrameID);
  disk_manager_->ReadPage(page_id, replPage->data_);
  latch_.unlock();
  return replPage;
}

if (!replacer_->Victim(&pageFrameID)) {
  latch_.unlock();
  return nullptr;
}
replPage = &pages_[pageFrameID];
```

If free_list_ contains some free frames, the first frame ID in the free_list_ will be picked to store the page. page_table_gets updated with the new mapping, page_id_ is set, pin count is increased, is_dirty_ flag for the page is set to fasle, and finally read in the actual data to store it in the page.

If free_list_ does not contain any free frame, then have replacer_ find victim to evict

Buffer Pool Manager (Cont.)

buffer_pool_manager.cpp :: FetchPageImpl()

```
  if (replPage->IsDirty()) {
    disk_manager_->WritePage(replPage->page_id_, replPage->data_);
  }
  page_table_.erase(page_table_.find(replPage->page_id_));
  page_table_.insert({page_id, pageFrameID});

  replPage->page_id_ = page_id;
  replPage->pin_count_ = 1;
  replPage->is_dirty_ = false;
  replacer_->Pin(pageFrameID);
  disk_manager_->ReadPage(page_id, replPage->data_);

  latch_.unlock();
  return replPage;
}
```

if the victim page is dirty, write it to the disk using disk_manager_'s WritePage interface. Erase the old mapping and insert a new mapping for the frame. Then, finally pin and read in the page from disk to buffer while updating page_id_, pin_count_, is_dirty_ flag, properly

Buffer Pool Manager (Cont.)

buffer_pool_manager.cpp :: UnpinPageImpl()

```
bool BufferPoolManager::UnpinPageImpl(page_id_t page_id, bool is_dirty) {

  latch_.lock();
  frame_id_t unpinned_frame_id = page_table_.find(page_id)->second;
  Page *unpinned_page = &pages_[unpinned_frame_id];

  if (unpinned_page->pin_count_ <= 0) {
    latch_.unlock();
    return false;
  }

  frame_id_t victim_frame = page_table_.find(page_id)->second;
  unpinned_page->pin_count_--;
  unpinned_page->is_dirty_ |= is_dirty;
  replacer_->Unpin(victim_frame);

  latch_.unlock();
  return true;
}
```

if pin_count_ is less than or equal to 0, we cannot unpin so return false

otherwise, we find the frame id using the given page_id and unpin it usin the frame id. Also, if the page has been modified, we set the flag for the unpinned_page so that it will be written to the disk before eviction

Buffer Pool Manager (Cont.)

buffer_pool_manager.cpp :: NewPageImpl()

```
Page *BufferPoolManager::NewPageImpl(page_id_t *page_id) {
  // 0.   Make sure you call DiskManager::AllocatePage!
  // 1.   If all the pages in the buffer pool are pinned, return nullptr.
  // 2.   Pick a victim page P from either the free list or the replacer. Always pick from the free list first.
  // 3.   Update P's metadata, zero out memory and add P to the page table.
  // 4.   Set the page ID output parameter. Return a pointer to P.

  latch_.lock();
  size_t free_list_size = BufferPoolManager::free_list_.size();
  size_t unpinned_non_free_pages = BufferPoolManager::replacer_->Size();
  if (free_list_size == 0 && unpinned_non_free_pages == 0) {
    latch_.unlock();
    return nullptr;
  }
```

This part of the code checks if there exist free space in the buffer. If not, it returns nullptr.

Buffer Pool Manager (Cont.)

buffer_pool_manager.cpp :: NewPageImpl()

```
frame_id_t pageFrameID;
Page *new_page;
page_id_t new_page_id;
if (!free_list_.empty()) {
  pageFrameID = free_list_.front();
  free_list_.pop_front();
  new_page_id = disk_manager_->AllocatePage();
  page_table_.insert({new_page_id, pageFrameID});
  new_page = &pages_[pageFrameID];
  new_page->page_id_ = new_page_id;
  new_page->pin_count_ = 1;
  replacer_->Pin(pageFrameID);
  *page_id = new_page_id;
  latch_.unlock();
  return new_page;
}

if (!replacer_->Victim(&pageFrameID)) {
  latch_.unlock();
  return nullptr;
}
Page *replPage = &pages_[pageFrameID];
```

This finds free space to store a new page from the free_list_ first. Also, call AllocatePage() method of disk_manager_ to allocate space in the disk and get the page id for the new page. Then, store the new page in the buffer, increases pin_count_ and pin the page as well as returns the pointer to the buffered page.

If we cannot find free space from the free_list_, we should find victim using replacer_'s Victim() method.

Buffer Pool Manager (Cont.)

buffer_pool_manager.cpp :: NewPageImpl()

```
  if (replPage->IsDirty()) {
    disk_manager_->WritePage(replPage->page_id_, replPage->data_);
  }
  page_table_.erase(page_table_.find(replPage->page_id_));
  new_page_id = disk_manager_->AllocatePage();
  page_table_.insert({new_page_id, pageFrameID});

  replPage->page_id_ = new_page_id;
  replPage->pin_count_ = 1;
  replacer_->Pin(pageFrameID);
  replPage->is_dirty_ = false;
  replPage->ResetMemory();

  *page_id = new_page_id;

  latch_.unlock();
  return replPage;

}
```

if the victim page is dirty, write it to the disk using disk_manager_'s WritePage interface. Erase the old mapping, allocate page, receives new page's id using disk_manager_'s AllocatePage() method, and insert a new mapping for the frame. Then, finally pin and reset the memory before returning the pointer to the newly allocated page.

Outline

- Buffer Pool Manager
- Assignment

Assignment: Finish up and Submit!

- Finish your implementation, test it, zip it and submit it on LMS as Lab2-submission.zip
  - You only need to include the following files with their full path in your submission zip file:
    - src/include/buffer/lru_replacer.h
    - src/buffer/lru_replacer.cpp
    - src/include/buffer/buffer_pool_manager.h
    - src/buffer/buffer_pool_manager.cpp
  - zip it as follows:

```
bhkimhy@bhkim-desktop:~/projects/advDB/bustub-private$ zip Lab2-submission.zip src/include/buffer/lru_replacer.h src/buffer/lru_replac
er.cpp src/include/buffer/buffer_pool_manager.h src/buffer/buffer_pool_manager.cpp
  adding: src/include/buffer/lru_replacer.h (deflated 57%)
  adding: src/buffer/lru_replacer.cpp (deflated 68%)
  adding: src/include/buffer/buffer_pool_manager.h (deflated 74%)
  adding: src/buffer/buffer_pool_manager.cpp (deflated 72%)
```

  - Verify it using unzip as follows:

```
bhkimhy@bhkim-desktop:~/projects/advDB/bustub-private$ unzip -l Lab2-submission.zip
Archive:  Lab2-submission.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     1217  2023-02-15 13:37   src/include/buffer/lru_replacer.h
     1507  2023-03-29 21:01   src/buffer/lru_replacer.cpp
     5869  2023-02-08 21:23   src/include/buffer/buffer_pool_manager.h
     5884  2023-02-18 22:13   src/buffer/buffer_pool_manager.cpp
---------                     -------
    14477                     4 files
```

The End