COMP 530 Assignment 1: LRU Buffer Manager

Due Wednesday, February 1st at 11:55 PM.

1. The Task

In Assignment 1, your task is to build a LRU buffer manager. This buffer manager is responsible for managing a pool of pages that are going to be used to buffer temporary data, and that will be used to buffer pages from database tables. This buffer manager will serve as the foundation that the rest of our database system (called "MyDB").

In the archive `A1.zip`, I've provided for you a skeleton of the project. You are required to use this skeleton—we'll be adding to this code base and extending it over the semester, so everyone has to be on the same page (no pun intended!).

In particular, for A1, you'll be extending the code in the files `MyDB_BufferManager.h`, `MyDB_PageHandle.h`, `MyDB_BufferManager.cc`, and `MyDB_PageHandle.cc`. Currently, the latter two files are just placeholders so that things compile. The former two contain the interface that you are required to implement, though you can add additional member functions and data members as needed. It's also likely the case that you'll be adding new helper classes that will help you to perform the various buffer management tasks.

2. Details, Details, Details

2.1 Types of Pages

There are two types of pages managed by this LRU buffer manager; pages that are associated with some position in an actual file/database table, and pages that are anonymous and are used as temporary storage by the rest of your system. Your buffer manager will have to write both kinds of pages back to disk to free memory when the buffer runs out of free pages. The difference between the two page types is that pages mapped to an actual file are written exactly to the file and position that they are mapped to, but anonymous pages are just stored somewhere on disk; the user of the buffer manager does not care where. In your implementation, all anonymous pages should be mapped to slots in a single temporary file.

2.2 Pinning Pages

Note that with the interface we've defined, it is possible to pin pages. A pinned page is always kept in RAM; it is never written back to disk. Thus, allocating a pinned page essentially takes a page out of the buffer pool, and the page's memory is not available to other pages until the user is done with the pinned page. Thus, the number of pinned pages cannot exceed the size of the buffer pool, since the pool can't have less than zero available memory.

2.3 Pages versus Page Handles

As you'll see when you look over the code skeleton, the buffer manager's interface returns handles to pages, and not pages themselves. The pages themselves remain under the control of the buffer manager and are not accessible to users of the buffer manager; the pages are accessed by the rest of the system via the handles.

When a user requests the same page multiple times, the page is only added to the pool one time. However, multiple handles to the page are created, one for each request.

When all of the handles to an anonymous page are gone, then it is impossible for the anonymous page to ever be accessed again, and the page's memory is automatically returned to the pool of unused memory, pinned or not. Since the page can never be accessed, the disk slot associated with the anonymous page should be recycled, so that subsequent anonymous pages can use that slot.

When all of the handles to a pinned, non-anonymous page are gone, then the page should be automatically unpinned. Later requests for that page may create new handles for the page, but at the point that the page has no more handles to it. Thus, it becomes possible for the page to be evicted from the buffer pool and its memory assigned to another page. Of course, the page can be re-pinned if, in the future, a handle to the same page is requested, but it is specified that the page should be pinned.

Since handles are implemented via C++ smart pointers, the destructor for the handle's class is automatically called when it is impossible to reach the handle (when the handle does not remain in any containers, and it has gone out of scope). At that point, you can reduce the reference count for the page.

2.4 Memory Allocation and File I/O

I'll ask you to allocate all of the buffer manager's memory all at once, in the constructor for the buffer manager. Memory for pages should not be allocated on-the-fly. You can either use `malloc` or `mmap` to do the allocation.

FIle I/O should be done using the `open`, `read`, `write`, and `lseek` libc functions. Make sure to use the `O_FSYNC` option when opening the file, so that writes are not buffered by the operating system.

2.5 Testing And Grading

In this class, we're using a very simple little test harness for C++ called `Qunit`. I have written one very extensive `Qunit` test for the buffer manager, and included it in the skeleton. It is located in `BufferTest/source/BufferQUnit.cc`. When you build your project using the `SCons` build tool (see below), this file will be complied and an executable will be produced.

If you can make it through my one big test case, it is likely that you've worked through the majority of the bugs in your buffer manager, and you've got most of the functionality done. However, you'll probably want to create a few additional test cases of your own—some simpler ones that you can use early on as you develop your code, and some additional, nasty ones just to make sure that everything is working.

When you turn in your code, and it's time for us to grade, we'll run the one big test case I've suppled, as well as several others that won't be made public until after the turnin. You'll be graded on your code's success in passing all of the test cases, though we revere the right to browse through your code and take off additional points if it appears you are missing some functionality or have somehow hacked something in a sketchy sort of way. You won't be graded on style and comments. However, I strongly encourage you to take this opportunity to put your best software engineering practices to use.

2.6 Project Difficulty

To let you know about how much code you'll write, here's the size of my own solution:

```
[cmj4@ring Main]$ wc -l BufferMgr/*/*.cc BufferMgr/*/*.h
  273 BufferMgr/source/MyDB_BufferManager.cc
   40 BufferMgr/source/MyDB_Page.cc
   17 BufferMgr/headers/CheckLRU.h
   97 BufferMgr/headers/MyDB_BufferManager.h
   79 BufferMgr/headers/MyDB_Page.h
   57 BufferMgr/headers/MyDB_PageHandle.h
   27 BufferMgr/headers/PageCompare.h
   27 BufferMgr/headers/TableCompare.h
  617 total
```

617 lines. That includes a lot of comments. Not huge, but not tiny. I'd anticipate that most teams are going to put in around 10 hours per person (teams of two) to get this assignment completed.

3. Getting Started

Lastly, I'm going to describe how to get started on the assignment. I'm going to give instructions on how to get started on Clear, *because it is required that everyone get their code to run on Clear.* Let me say this again: **to get *any* credit on A1, your code must compile and run on Clear**. That way, we have a common environment for grading and we don't have to spend time getting your code to compile. That said, I suspect that almost everyone is going to do the assignment on their own laptop or home computer, and then copy over to clear to verify that everything works right before submission. Which is fine.

You will likely need to modify this process at least somewhat to get things to work in your own environment.

First you need to install the `SCons` build tool. `SCons` is a modern build tool (sort of like make, but far more useful because it is Python-based, and it allows you to include scripting code within your build tool). We'll be using `SCons` throughout the class.

Clear does not have `SCons` installed. So in your home directory, download `SCons`:

```
[cmj4@glass bin]$ wget --no-check-certificate https://pypi.python.org/
packages/source/S/SCons/scons-2.4.1.tar.gz
```

Unpack it:

```
[cmj4@glass ~]$ gunzip scons-2.4.1.tar.gz
[cmj4@glass ~]$ tar xvf scons-2.4.1.tar
```

Build it:

```
[cmj4@glass ~]$ mkdir scons
[cmj4@glass ~]$ cd scons-2.4.1/
[cmj4@glass ~/scons-2.4.1]$ python setup.py install --prefix=../scons
[cmj4@glass ~]$ cd ..
[cmj4@glass ~]$ rm -r scons-2.4.1/
```

And now you can run it (though nothing is going to happen because there is no `SConstruct` file in this directory to tell `SCons` what to do):

```
[cmj4@glass ~]$ ~/scons/bin/scons-2.4.1
```

Next, download the starter `A1.zip` file from OwlSpace, and unzip it. Then go into the build directory and use `SCons` to build the project:

```
[cmj4@glass ~]$ cd A1/Build/
[cmj4@ring Build]$ ~/scons/bin/scons-2.4.1
scons: Reading SConscript files ...

What do you want to build/clean?

1. Buffer unit tests
2. Buffer unit tests for Clear (use clang++ compiler)

Select the module(s) you want to build or clean. 2

OK, building buffer unit tests using clang++.
scons: done reading SConscript files.

scons: warning: Support for pre-2.7.0 Python version (2.6.6) is
```

```
deprecated.
    If this will cause hardship, contact scons-dev@scons.org
File "/storage-home/c/cmj4/scons/bin/scons-2.4.1", line 199, in
<module>
scons: Building targets ...
```

Once the build completes, you can run the code. Since the methods are all empty, the program will crash:

```
[cmj4@ring Build]$ bin/bufferUnitTest
allocating pinned page
Segmentation fault (core dumped)
```

At this point you are ready to begin work on the project. Note that as long as you keep your header and source files in the directories that are currently there, `SCons` will automatically find and build them.

4. Turnin

Simply zip up all of your source code and then turn it in on OwlSpace (make sure to archive into the zip format, and not some other archiving format. If you choose to use something else, we'll take off a few points!). Please name your archive A1.zip. Please do not change the original directory structure, except for perhaps adding some new files. The root should be a directory called A1, with two subdirectories Build and Main. And so on.

And remember, this needs to compile and run on Clear.

Finally, and this is important: **include a README file in the root of your project with any important information**, including the names of the one or two people who worked on the project.

And also: if you work with a partner, **only turn in one copy of your source**. Otherwise, we'll possibly grade your submission twice, and end up getting quite annoyed.

Good luck and have fun!