UNIVERSITY OF CALIFORNIA
College of Engineering
Department of EECS, Computer Science Division

Assignment 1

Prof. Joe Hellerstein
Dr. Minos Garofalakis

Assignment 1: PostgreSQL 8.0.3 Buffer Manager

In the previous assignment, we learned how to set up a database cluster (using initdb), how to start the postgres master process (using pg_ctl), how to create a specific database (using createdb), and how to query this database (using a front end tool like pgaccess or psql).

Now we start with the actual C code hacking. The aim is to modify functionality in the backend server of PostgreSQL. For this first assignment, we focus on just one core module - the *buffer manager*. The actual coding for this assignment is minimal, given the highly organized and modular PostgreSQL code (the code for the buffer manager policy is cleanly separated from the rest of the code base). However, you have to figure out what code to modify, which is non-trivial! There are three major parts to this project:

- Understanding different buffer management strategies
- Examining, understanding and modifying existing code
- Testing your code in a real system

Partners?

There are two parts to this assignment, the first of which is to be done individually and the second in teams of 2 people. For this, you will work in groups of 2. Please form a your team of 2 and register at the group registration webpage by **Friday Sept. 9**.

Deadlines

You have to submit this assignment in **TWO PHASES**:

1. The first part will test your understanding of different buffer replacement strategies and is due on **Tuesday Sept 13 at 11:59pm**. This part must be done individually.
2. The second phase requires that you and your partner code a buffer replacement policy and create tests code. The due date for the part is **Tuesday Sept. 20 at 11:59pm**.

The second part is expected to take much more time than the first. So manage your time accordingly!
The rest of this document describes your tasks in each phase.

Tasks and Grade Percentage

1. Understanding different buffer replacement strategies. (20%)
2. Implement the CLOCK buffer replacement strategy. (80%)
3. Add test code using the provided test harness. (0%)

The rest of this document describes these steps in detail. Note that in the code and in this document a "buffer" or "slot" is what the text and course notes refer as a "buffer frame". Similarly, the terms "refcount" and "pincount" are used interchangeably.

1. Understanding Different Buffer Replacement Strategies. (20%)

The textbook describes LRU, MRU and CLOCK strategies in section 9.4.1. However, you may need to read the entire section (9.4) to get a full image of a buffer manager in the DBMS. A fourth (more recent) strategy is called 2Q, and is used in the latest version of Postgres (8.0.3). LRU, MRU, and CLOCK strategies work well enough for most systems so we are going to replace 2Q with a CLOCK strategy. But first you'll need to exercise your understanding of the LRU, MRU, and CLOCK buffer replacement strategies.

This part of the assignment requires no code. We will give you the number of page slots that the buffer manager must manage and a sequence of data blocks accessed by the different DBMS layers above the buffer manager layer. You will have to track the behavior of the buffer manager (which pages it has to replace, when, etc.) using your, and yours alone, knowledge of the three aforementioned buffer replacement strategies.

Solution format for this step: You must record your answers in three plain text files called *strategy.lru*, *strategy.mru*, *strategy.clock*. These files will contain the buffer/slot state and buffer manager replacement decisions in accordance to the LRU, MRU, and CLOCK policies respectively. The format of each file is as follows:

- Three columns of text, separated by spaces
- One line of text each time a page miss occurs
- Each row lists the time of a page miss, a space, the buffer pool page (P1, P2, P3, P4), a space and, if an eviction occurs, the page evicted from memory

For example, assume there are four page slots your buffer manager must manage: P1, P2, P3, P4. All four slots are initially empty. When the buffer pool has unused slots (e.g., in the beginning when all four slots are empty), it will put the newly read data in the first unused slot. The blocks to be read from disk are labeled A through F. For simplicity here, we assume that after each access the page is pinned, and then immediately unpinned (you cannot make this same assumption when writing the actual code. A page may be pinned for any length of time in a DBMS!)

Given this information and the following workload:

Time	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
Block Accessed	A	A	B	C	D	E	F	A	B	C	D

the files should be as follows:

strategy.lru
T1 P1
T3 P2
T4 P3
T5 P4
T6 P1 A
T7 P2 B
T8 P3 C
T9 P4 D
T10 P1 E
T11 P2 F

strategy.mru
T1 P1
T3 P2
T4 P3
T5 P4
T6 P4 D
T7 P4 E
T11 P3 C

strategy.clock
T1 P1
T3 P2
T4 P3
T5 P4
T6 P1 A
T7 P2 B
T8 P3 C
T9 P4 D
T10 P1 E
T11 P2 F

Note that certain times (e.g., T2) are missing for each strategy. This is due to the different buffer miss patterns exhibited by the given replacement policy. Note also that the third column will be blank when the buffer miss does not result in an eviction. Finally, observe that in this case, for the given number of pages and sequence of block accesses, MRU emerges the winner (least number of misses).

Now we come to the problem that you will have to solve. Suppose the buffer manager has five page slots to manage: P1, P2, P3, P4, P5. All slots are initially empty. Suppose seven disk blocks (A, B, ..., G) are accessed by the layers above the buffer manager layer in the DBMS.

To access your access pattern, login to your cs186 account and type **mypattern** at the prompt. Remember that you must use your cs186 class account to get your access pattern. Based on this information, you should perform a similar analysis as the one above.

Generate the three files strategy.lru, strategy.mru, and strategy.clock for this problem. Stick to the specified format (e.g., no extra line at the end of a file), as we will use file comparison scripts to evaluate your answers.

How to submit: You will need to use the unix **submit** program to hand in your assignment:

1. Save your three files in a directory called "hw1p1" within your cs186 home directory.
2. cd into hw1p1.
3. Run: **submit hw1p1**

3. Implementing the CLOCK Buffer Replacement Strategy. (80%)

3.1. Setting up PostgreSQL and Other Files/Scripts

Everything needed for the second part of the assignment is available at: ~cs186/Hw1/ on x86 solaris instructional machines. Because the source tree of Postgres is quite large, you'll need to delete the database you have setup for homework 0 before you begin. So start by:

```
/usr/bin/rm -rf ~/pgdata
```

Be careful! This command will not prompt you for delete confirmation.
Once you have removed homework 0's database, you can un-tarzip the file:

```
cp -r ~cs186/Hw1 ~/.
```

This creates a Hw1 directory in your top level home directory. Inside that directory, you will see two files: **setup.py** and **config.in**. The python script setup.py does precisely what its name suggests, so go ahead and execute it (./setup.py) on any instructional machines (basically, if setup.py runs then you can use that machine). After the script completes, you should see the following directory structure setup for you.

1. **postgresql-8.0.3/:** Source code of PostgreSQL version 8.0.3. Little code was changed between this version and the actual version.

2. **MyStuff/:** This is the directory in which you will spend most of your time. It has a couple of subdirectories:

   - **MyCode/:** In this directory you will find **freelist.c** and **freelist.skel.c**. The file freelist.c holds the buffer policy used by PostgreSQL, while freelist.skel.c is skeleton code that you will need to fill in. Again, PostgreSQL ships with 2Q, and can serve as example code for each of the routines required by the freelist interface. HINT: Focus on the comments in the freelist.skel.c skeleton code. They will tell you the purpose of a particular routine along with a high level description of what the code should look like.

   - **PerformanceEvaluationScripts/:** This has the scripts that you will use to carry out PostgreSQL experiments after you have written completed freelist.skel.c and compiled it successfully. The arguments to the provided shell scripts are self explanatory (all you need to do is supply the location of where you want your data directory to be placed e.g., DoInitDB.sh ./data, PrepareTables.sh ./data, Query.sh ./data). The Query.sh shell script will execute a query, producing output to stdout.

Here are instructions for compiling PostgreSQL and incorporating freelist.skel.c into PostgreSQL. Note that compiling/updating may take a few minutes.

- To completely re-compile and install PostgreSQL,

```
cd Hw1/postgresql-8.0.3/
gmake            # Simply compiles the entire PostgreSQL source tree
gmake install    # Default: install to location $HOME/pgsql
```

- After you have prepared your version of freelist.skel.c, and have tested it using the test harness code, you will need to install it in the PostgreSQL source tree. The path to the buffer manager policy is **postgresql-8.0.3/src/backend/storage/buffer/freelist.c**, which you will replace with your version of freelist.skel.c. Note: Save a copy of the 2Q version in case you need it in the future.

```
cp freelist.skel.c ~/Hw1/postgresql-8.0.3/src/backend/storage/buffer/freelist.c
cd ~/Hw1/postgresql-8.0.3/
gmake; gmake install;   # Compile and install
```

NOTE: If you're working on the instructional machines and need to kill you postmaster instance (due to it not shutting down properly) you can use the **kill_postmaster** command. In most cases you should be able to cleanly shutdown using pg_ctl -D <data directory> stop. Also, we STRONGLY recommend that you do not modify any other files in the postgresql-8.0.3 directory! Moreover, we suggest you work from your /MyCode directory and replace freelist.c in src/backend/storage/buffer/ only when you need to test your freelist.skel.c changes.

3.1.1 Working from a local machine

A tar gzipped file containing the PostgreSQL 8.0.3 code base can be found here. This should provide you with the necessary files to complete this assignment using whatever machine you'd like (assuming PostgreSQL 8.0.3 compiles). We will not support platform issues concerning this code base (I have successfully tested it using Linux and OS X).

3.2 Implementing CLOCK

The description of the **CLOCK** algorithm is available in the course textbook (Section 9.4). We give a brief description in the context of PostgreSQL to assist with your implementation.

To implement the CLOCK replacement policy, you'll need to maintain (but are not limited to) the following information.

1. A referenced bit associated with each page. Each time a page in the buffer pool is unpinned, the referenced bit associated with the corresponding frame is set to 1 before the page is considered for replacement. We have done this step for you by adding such a reference variable to the BufferDesc data structure titled “reference” (see src/include/storage/buf_internals.h), which is set to “true” after each **bufmgr.c Unpin(…)** page operation (see src/backend/storage/buffer/bufmgr.c). You will use this reference variable when making your replacement decision.
2. The “clock hand” varies between 0 and NBuffers - 1, where NBuffers is the number of buffers in the PostgreSQL buffer pool. Each time a page needs to be found for replacement, the search begins from the page pointed to by the clock hand and advances to current = (current + 1) % NBuffers if not found (i.e. in a clockwise cycle). We have supplied you with a data structure in freelist.skel.c named BufferStrategyControl containing a field named clock_position.

Each page in the buffer pool is in one of three states:

Pinned	BufferDescriptors.refcount > 0
unpinned AND referenced	BufferDescriptors.refcount = 0 AND BufferDescriptors.referenced = true
unpinned AND not referenced (available)	BufferDescriptors.refcount = 0 AND BufferDescriptors.referenced = 0

To find a page for replacement you start from the current page (initialized with the current clock hand), and repeat the following steps

until an available page is found.

page[current] is pinned	advance current and try again
page[current] is unpinned and referenced	set reference to false and advance current
page[current] is unpinned and not referenced	use this page as replacement victim

3.2.1. Files of interest

You can add and manage any new data structures that you need. The existing code is not extremely clear, but it is understandable. It may take you a few hours (or more) to digest it. Since understanding the existing code is a significant part of the assignment, the TAs and Professors will not assist you in your understanding of the code base (beyond what we discuss here).

The actual buffer manager code is neatly separated from the rest of the code base. It is located in the postgres-8.0.3/src/backend/storage/buffer directory, and primarily made up of the files bufmgr.c and freelist.c. While bufmgr.c contains the implementation of the buffer manager, we are only interested in freelist.c, which defines the buffer manager strategy (e.g., LRU, MRU, CLOCK, etc.).

- **src/backend/storage/buffer/freelist.c** - has functions that implement the replacement strategy. This is the only file you need to replace using your filled in freelist.skel.c.
- **src/include/storage/buf_internals.h** - contains the definition of each buffer frame (called BufferDesc). Most of the fields in BufferDesc are managed by other parts of the code. The fields that you'll be interested in are the BufferDesc.refcount and BufferDesc.reference. The **reference** field is set to **true** by the buffer manager when the **refcount** (a.k.a. pin count) transitions from 1 to 0. You should convince yourself that this yields the proper semantics needed to successfully execute the CLOCK algorithm.
- **src/backend/storage/buffer/buf_init.c** - Some initialization of the buffer frames occur in this file. However, you should do all your initialization in freelist.c (see StrategyInitialize routine in freelist.c).

3.2.2. How to debug?

We suggest that you debug using gdb or ddd with the src/backend/postgres standalone executable. Make sure you run your version of postgres and not the one in the class account directory! To start the backend server in stand-alone mode, run **src/backend/postgres -s -D <DATADIR> test**. It will use the data directory (and data) you prepared using **initdb -D <DATADIR>** and database test you created using **createdb test**. The interface is directly to the backend, so psql features will not work. If your binary does not work correctly, it may corrupt the data, so be warned.

Through the backend, you can directly type SQL statements, although the output is somewhat painful to read. To exit, type **CTRL-D**. In addition, the **-s** option will tell the server to print statistics at the end of each query, including the number of buffer hits! Finally, you can restrict the number of buffers PostgreSQL will use by adding the **-B <nbuffers>** option (<nbuffers> should not be smaller than 16.) so that even small queries generate buffer misses.

More information on using the backend in stand-alone executable can be found on the PostgreSQL web site: http://www.postgresql.org/. We will provide a C Programming help session (see course blog for this announcement) that will cover debugging in PostgreSQL.

Other debugging options include the use of debugging output using the following routine:

```
elog(DEBUG,<format>,<arg>)
```

Please see bufmgr.c for example uses of this routine.

4. Test harness code. (0%)

A small test harness stub program can be found in Hw1/MyStuff/MyCode/buftest.c. This stub file tests the correctness of your buffer policy (freelist.c) by bypassing the PostgreSQL code and directly testing your freelist.c implementation.

Copy your version of freelist.skel.c to freelist.c and run ***gmake*** to compile the test harness stub. Full details about using the test stub are available in a README file within the /MyCode directory.

For the final task of this assignment, you will create buffer access patterns in the buftest.c file (part 1 of this assignment should assist you in this effort). The goal is to add access patterns that ensure your code provides the correct functionality (e.g., runs CLOCK algorithm, returns unpinned and unreferenced pages on request, etc.). You will receive no credit for this part, but we ask that you submit your buftest.c anyway. It should be clear from looking at the code in buftest.c, how to go about writing extra test cases. We strongly recommend that you make sure the test harness runs properly on your code before installing it into PostgreSQL. A working test harness does not imply a working version of your buffer manager within the confines of PostgreSQL.

Please be sure to submit both buftest.c and freelist.c in the final submission. Again we emphasize, there is no credit for this part (so you don't have to do anything with buftest.c if you don't want) but adding (sound) tests to buftest.c will verify the correctness of your buffer manager.

5. Final submission.

How to submit: You will need to use the unix **submit** program to hand in your assignment:

1. Save your freelist.c and buftest.c files in a directory called "hw1p2" in your cs186 home directory.
2. cd into hw1p2.
3. Run: **submit hw1p2**

Warning: Compilation errors in freelist.c or buftest.c **will be** harmful to your credit on the respective parts. If your code files depend on changes that you made to other files (not freelist.c or buftest.c), then you should redesign your solution without such modifications.