National University of Singapore
School of Computing

CS3223: Database Systems Implementation
Semester 2, 2022/23

Assignment 1 (10 marks)

This assignment consists of two parts with different deadlines.

Part 1 (2 marks) is an individual assignment which is due on February 12 (Sunday 2359).

Part 2 (8 marks) is a team assignment which is due on February 22 (Wednesday 2359).

Late submission penalty: For assignment part 2, there will be a late submission penalty of 1 mark per day up to a maximum of 3 days. If your assignment is late by more than 3 days, it will not be graded and your team will receive 0 marks for part 2. Do start working on your assignment early!

1 Using SoC's Linux Servers

This assignment will be done using SoC's Linux servers. You may also complete this assignment on a Windows/macOS PC, but you will have to figure it out on your own the required additional software and steps; some guidelines are given in Appendix A.

To access SoC's Linux servers, you will need to have a SoC account. If you do not have a SoC account, visit https://mysoc.nus.edu.sg/~newacct/ to apply for one. Once you have an account, visit https://mysoc.nus.edu.sg/~myacct/services.cgi to activate your access to SoC's servers. When your account is activated, you may ssh to SoC's linux server for students (known as stu) at **stu.comp.nus.edu.sg**.

Other Linux servers that you may use are **xlog0.comp.nus.edu.sg**, **xlog1.comp.nus.edu.sg**, and **xlog2.comp.nus.edu.sg**.

To login to the stu server from your PC, you need to have a ssh client installed on your PC. Linux and macOS users should already have a ssh client installed by default. Windows users could install the Windows Subsystem for Linux (WSL) and use Linux. You could also use other ssh clients such as PuTTY.

To access SoC's servers from a machine that is outside of SoC's internal network, you will need to go through SoC VPN. Note that SoC VPN is different from NUS VPN. Connecting to NUS VPN only allows you access to the NUS internal network, but not the SoC internal network.

2 Getting Started

First, ssh to the stu.comp.nus.edu.sg server using your SoC userid (e.g., ssh alice@stu.comp.nus.edu.sg). In the rest this document, assume that "$" is the the command prompt of your login shell, and the commands that are entered are shown in blue.

Run the following commands to download the assignment files.

```
$ cd $HOME
$ wget http://www.comp.nus.edu.sg/~cs3223/assign/cs3223_assign1.zip
$ unzip cs3223_assign1.zip
$ cd cs3223_assign1
$ ls
```

The `cs3223_assign1` sub-directory created in your home directory contains the following directories and files: Makefile, four C programs (bufmgr.original.c, freelist.original.c, bufmgr.c, freelist-lru.c), four bash scripts (install.sh, part1.sh, part2.sh, test.sh), test_bufmgr/, testdata/, and testresults-lru-solution/.

3 Part 1: Compiling PostgreSQL (2 marks)

In this assignment part, you will learn how to compile PostgreSQL (version 15.0) on a Linux server, and run a benchmark test on PostgreSQL.

First, run the **install.sh** script to install PostgreSQL 15.0 from its source files. This script assumes that you have already unzipped the assignment zip file to create the directory ~/cs3223_assign1/. The PostgreSQL source files will be downloaded into the directory ~/postgresql-15.0, PostgreSQL will be installed in the directory ~/pgsql, and the database storage files will be stored in the directory ~/pgdata. The above directories can be configured by modifying the appropriate variables in the install.sh script. The script also configures the environment variables PATH, MANPATH, PGDATA, and PGUSER in ~/.bash_profile

Note that the compilation process may take a while.

```
$ cd cs3223_assign1
$ ./install.sh
```

Next, start the PostgreSQL server and execute the **part1.sh** script to run a benchmark test on PostgreSQL as follows. The source command executes the modified .bash_profile file to set up the modified environment variables.

The pg_ctl start command starts the PostgreSQL server. The -l option specifies the file (e.g., log.txt) to store the log messages from the PostgreSQL server.

By default, the PostgreSQL server listens on the port number 5432 for client connections. However, since there are likely to be many instances of PostgreSQL server running on the Linux machine at the same time, you should choose an unused port number when starting the server. The port number is configured using the -p option; in the example shown below, port number 5001 was used. If you see the error message ‘‘pg_ctl: could not start server’’ when trying to start the server, you should check the log file to determine the reason of the failure; if you find the error message ‘‘FATAL: lock file "postmaster.pid" already exists’’, it means that the port number that you were using is already in use. To see which port numbers are currently being used, run the command `ls -a /tmp/.s.PGSQL.*`; for example, if the ls command shows the file /tmp/.s.PGSQL.5432, it means that the port number 5432 is currently being used by another server instance. For convenience, it is recommended for each team to use the port number 5nnn where *nnn* is your team number.

The -B option specifies the number of shared buffers to be used by the server. For the benchmark test, you will use a buffer pool with 8192 buffer frames.

```
$ source ~/.bash_profile
$ pg_ctl start -l log.txt -o "-p 5001 -B 8192"
$ export PGPORT=5001
$ ./part1.sh
$ pg_ctl stop
```

Before running the part1.sh script to start the benchmark test, it is important to exceute the **export** command to set the PGPORT environment variable to the port number used in the previous pg_ctl start command.

The **part1.sh** script runs a simple benchmark test by simulating 10 clients issuing transactions on a synthetic four-relation database for a duration of 3 minutes. The benchmark results are stored in the file named **part1_result.txt**.

The following performance metrics are reported by the benchmarking:

- average transaction latency (in milliseconds),
- system throughput (in number of transactions processed per second), and
- buffer hit ratio which measures how often a requested data page is found in the buffer pool without incurring disk I/O to read the page from the disk; i.e., it is the ratio of the total number of page requests that are found in the buffer pool to the total number of page requests.

Finally, the **pg_ctl stop** command stops the server. You should stop your PostgreSQL server before you logout.

3.1 Part 1: What & How to Submit

Rename the file **part1_result.txt** to a name of the form eNNNNNNN_part1.txt, where eNNNNNNN is your NUSNET account id (e.g., e0123456_part1.txt).

```
$ mv part1_result.txt e0123456_part1.txt
```

Use the **sftp** command to transfer the renamed file from the stu server to your PC.

```
sftp alice@stu.comp.nus.edu.sg
alice@stu.comp.nus.edu.sg's password:
sftp> cd cs3223_assign1
sftp> get e0123456_part1.txt
sftp> quit
```

Upload the renamed text file to **Assignment 1 Part 1** on Canvas. The deadline for this submission is February 12 (Sunday 2359). As this is an indvidual assignment, each student must make your own individual submission.

4 Part 2: LRU Replacement Policy (8 marks)

PostgreSQL uses a variant of the Clock policy as the default buffer replacement policy.
In this assignment part, you will modify the buffer manager component of PostgreSQL to implement the LRU replacement policy.

4.1 Overview of PostgreSQL

The following is a brief introduction to key aspects of PostgreSQL that are relevant for this assignment.

4.2 Shared-memory Data Structures

In general, there could be multiple backend server processes accessing a database at the same time. Therefore, access to shared-memory structures (e.g., buffer pool data structures) needs to be controlled to ensure consistent access and updates. To achieve this, PostgreSQL uses *locks* (e.g., spin locks, light-weight locks) to control access to shared-memory structures by following a locking protocol: before accessing a shared-memory structure, a process needs to acquire a lock for that structure; and upon completion of the access, the process needs to release the acquired lock.

4.3 Buffer Manager

Thre are two main types of buffers used in PostgreSQL: a *shared buffer* is used for holding a page from a globally accessible relation, while a *local buffer* is used for holding a page from a temporary relation that is locally accessible to a specific process. This assignment is about the management of PostgreSQL's shared buffers.

Initially, all the shared buffers managed by PostgreSQL are maintained in a *free list*. A buffer is in the free list if its contents are invalid and is therefore of no use to any client process. Whenever a new buffer is needed, PostgreSQL will first check if a buffer is available from its free list. If so, a buffer from the free list is returned to satisfy the buffer request; otherwise, PostgreSQL will use its buffer replacement policy to select a victim buffer for eviction to make room for the new request. PostgreSQL 15.0 implements a variant of the Clock algorithm as its default replacement policy.

In PostgreSQL, when a record is deleted (or modified), it is not physically removed (or changed) immediately; instead, PostgreSQL maintains multiple versions of a record to support a *multiversion concurrency control* protocol (to be covered in the later part of the course). Periodically, a *vacuuming* process will be run to remove obsolete versions of records that can be safely deleted from relations. If it happens that an entire page of records have been removed as part of this vacuuming procedure, the buffer holding this page becomes invalid and is returned to the free list. PostgreSQL also runs a background writer process (called bgwriter) that writes out dirty shared buffers to partly help speed up buffer replacement.

In PostgreSQL, the buffer pool is implemented as an array of disk blocks, where the index to each array entry is referred to as a **buffer_id**, and each disk block's location is identified by a **buffer tag**. The pin count for each buffer frame is known as **reference count (or refcount)**. Each buffer frame is also associated with a **buffer descriptor** which stores metadata about its contents. Given a buffer tag, a hash-based **buffer table** is used to efficiently locate the buffer id of the buffer frame that stores the disk block (corresponding to the given buffer tag) if the disk block is resident in the buffer pool.

PostgreSQL's buffer manager implementation uses various spin locks and light-weight locks to control access to its shared-memory data structures. For example, the metadata for the Clock replacement policy is stored in a data structure pointed to by a global variable named *StrategyControl* which consists of a spin lock named *buffer_strategy_lock*. This spin lock is used to control access to various data structures (e.g., free list) by using the functions *SpinLockAcquire* and *SpinLockRelease*, respectively, to acquire and release the spin lock *buffer_strategy_lock*.

4.4 Implementing LRU Strategy

A simple approach to implement the LRU policy is to use a doubly-linked list to link up the buffer pages such that a page that is closer to the front of the list is more recently used than a page that is closer to the tail of the list. Whenever a buffer page is referenced, it is moved to the front of the list; and whenever a replacement page is sought from the list, the unpinned buffer page that is closest to the tail of the list is selected for eviction. This implementation approach has been referred to as the *Stack LRU* method; here, the top and bottom of the stack correspond to the front and tail of the linked list, respectively. Whenever a buffer is accessed, its position within the stack needs to be adjusted. This stack adjustment can be classified into four cases:

(C1) If an accessed page is already in the buffer pool, then the containing buffer needs to be moved to the top of the stack.

(C2) If an accessed page is not in the buffer pool and a free buffer is available to hold this page, then the selected buffer from the free list needs to be inserted onto the top of the stack.

(C3) If an accessed page is not in the buffer pool and the free list is empty, then the selected victim buffer needs to be moved from its current stack position to the top of the stack.

(C4) If a buffer in the buffer pool is returned to the free list, then the buffer needs to be removed from the stack.

4.5 Implementation Guidelines

The following are some guidelines on how you can go about implementing the LRU replacement policy in PostgreSQL.

1. Before you begin making changes to PostgreSQL, you should examine the existing code to understand how the buffer manager (specifically its Clock policy replacement) is being implemented. 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 Professor 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. Its files are located in the src/backend/storage/buffer/ and src/include/storage/ directories. The two main files of interest for this assignment are bufmgr.c and freelist.c. Modified versions of these two files (to be used for this assignment) are given in cs3223_assign1/bufmgr.c and cs3223_assign1/freelist-lru.c. While bufmgr.c contains the implementation of the buffer manager, we are only interested in freelist-lru.c, which implements the buffer replacement policy.

   To identify the parts in cs3223_assign1/bufmgr.c and cs3223_assign1/freelist-lru.c that have been modified, search for the string ''cs3223''.

   The following is a brief description of some of the relevant files for this assignment:

   - cs3223_assign1/freelist-lru.c: Contains functions that implement the replacement strategy. This is the only file you need to modify.
   - cs3223_assign1/bufmgr.c: This is a modified version of src/backend/storage/buffer/bufmgr.c that implements the buffer manager to be used for this assignment.
   - src/include/storage/buf_internals.h: Contains the definition of the buffer descriptor (*BufferDesc*). Most of the fields in *BufferDesc* are managed by other parts of the code.
   - src/backend/storage/buffer/buf_init.c: Some initialization of the buffer frames occur in this file. However, you should do all your initialization (e.g., LRU-related data structures) in the StrategyInitialize function in freelist-lru.c.
   - src/backend/storage/buffer/README: Useful description of the *Strategy* interface implemented by freelist-lru.c.

2. For this assignment, the only file that you need to modify is cs3223_assign1/freelist-lru.c. The given file is a slightly modified version of the original file src/backend/storage/buffer/freelist.c. You will need to make further modifications to this file to implement LRU (instead of Clock) replacement policy. To help you with the LRU implementation, we have defined a new function in freelist-lru.c:

   ```
   void StrategyUpdateAccessedBuffer (int buf_id, bool delete)
   ```

   This function is used to update the LRU stack when a buffer, which is uniquely identified by its buffer index number given by buf_id, is accessed. The delete parameter is used to distinguish case C4 from the other three cases: the value of delete is *true* for handling case C4; and *false*, otherwise. This new function is used in two files: bufmgr.c, and freelist-lru.c. You will need to implement this new function. Refer to the BufferAlloc function in cs3223_assign1/bufmgr.c for an example of how StrategyUpdateAccessedBuffer is being used to update the LRU stack to handle case C1 (i.e., when the accessed page is already in the buffer pool).

3. Besides implementing the StrategyUpdateAccessedBuffer function, you will need to make other necessary changes to freelist-lru.c. You will probably need to modify the following functions in freelist-lru.c:

   - **StrategyInitialize** to allocate and initialize shared memory for your LRU-related data structures using ShmemInitStruct function (and not using function such as malloc),
   - **StrategyShmemSize** to account for any additional shared memory used by your LRU-related data structures using functions such as add_size, mul_size, etc.,
   - **StrategyGetBuffer** to handle cases C2 and C3, and
   - **StrategyFreeBuffer** to handle case C4.

You can use the cs3223_assign1/Makefile to compile your freelist-lru.c and install the modified files into the PostgreSQL source tree. Edit the Makefile, if necessary, to ensure that the SRC_DIR variable is correctly set to the directory path of your PostgreSQL source tree. To compile your freelist-lru.c, use the command ''make freelist-lru.o''.

4.6 Testing LRU Implementation

Once your freelist-lru.c compiles correctly, you are now ready to install your LRU implementation into the PostgreSQL source tree and test your implementation.

To install your changes, you need to replace src/backend/storage/buffer/freelist.c and src/backend/storage/buffer/bufmgr.c with cs3223_assign1/freelist-lru.c and cs3223_assign1/bufmgr.c, respectively, and re-install PostgreSQL. This can be done by simply excuting the command make lru[1].

```
$ cd ~/cs3223_assign1
$ make lru
```

Once your LRU implementation is installed successfully, test your implementation with the script **test.sh** by starting the server with 16 buffer pages. As before, you should modify the port number as appropriate in the following commands.

[1] To restore the original PostgreSQL installation, run the command "make clock".

```
$ pg_ctl stop
$ pg_ctl start -l log.txt -o "-p 5001 -B 16"
$ export PGPORT=5001
$ cd ~/cs3223_assign1
$ ./test.sh
$ pg_ctl stop
```

The testing comprises of 10 test cases (~/cs3223_assign1/test_bufmgr/testcases) that is part of the test_bufmgr extension installed by the install.sh script. Each test case is a sequence of read/unpin requests to pages of a relation named movies (~/cs3223_assign1/testdata) which occupies 43 8KB-pages (numbered from 0 to 42). Specifically, there are three types of page requests in the test cases:

- **read_pin_block(blkno)** reads the page with block number blkno and the page remains pinned;
- **read_unpin_block(blkno)** reads the page with block number blkno and unpins the page; and
- **unpin_block(blkno)** unpins the page with block number blkno.

The results of running the test cases are stored in the directory ~/**cs3223_assign1/testresults**. You can compare your results (e.g., using the diff command) against the provided files in ~/cs3223_assign1/testresults-lru-solution/

4.7 Benchmarking LRU Implementation

After testing and debugging your implementation of LRU policy, benchmark its performance by executing the following commands; you should replace the port number following the suggestion in Section 3.

```
$ pg_ctl stop
$ pg_ctl start -l log.txt -o "-p 5001 -B 8192"
$ export PGPORT=5001
$ ./part2.sh
$ pg_ctl stop
```

Similar to part1.sh script, the **part2.sh** script runs a simple benchmark test by simulating 10 clients issuing transactions on a synthetic four-relation database for a duration of 3 minutes. The benchmark results are stored in the file named **part2_result.txt**.

4.8 Part 2: What & How to Submit

For this assignment part, you should submit the two files, **part2_result.txt** and **freelist-lru.c**, by copying these files into a directory with a name of the form teamN, where N is your assignment team number (a number between 1 and 116). and compressing the directory into a zip file as illustrated by the following commands.

```
$ cd ~/cs3223_assign1
$ mkdir team1
$ cp freelist-lru.c part1_result.txt team1
$ zip -r team1.zip team1
```

Use the **sftp** command to transfer the zip file from the stu server to your PC.

```
sftp alice@stu.comp.nus.edu.sg
alice@stu.comp.nus.edu.sg's password:
sftp> cd cs3223_assign1
sftp> get team1.zip
sftp> quit
```

Upload the zip file to **Assignment 1 Part 2** on Canvas. The deadline for this submission is February 22 (Wednesday 2359). Each team should submit only one submission.

5 Resources

- PostgreSQL Documentation
- SoC's computer cluster
  - SoC cluster hardware configuration
  - SoC account policies
- Learning Linux
  - Getting Started with Linux
  - The Linux Command Line
  - UNIX Tutorial for Beginners
  - The Missing Semester of Your CS Education
- Learning C
  - Prof. Ooi Wei Tsang's CS1010 Notes
  - Online book on Modern C

A Compiling on MacOS/Windows PC

If you wish to compile PostgreSQL on your MacOS/Windows PC, you may refer to PostgreSQL's notes.

For Windows users, another option is to install Ubuntu Linux on **Windows Subsystem for Linux (WSL)**. For more information, you may refer to the following two references:

- Install Ubuntu on WSL2

- Install Linux on Windows with WSL

After Ubuntu Linux is installed, additional packages need to be installed using the command:
`sudo apt install bison build-essential flex gcc gdb libreadline-dev libssl-dev libxml2-dev libxml2-utils libxslt1-dev xsltproc zlib1g-dev`