Lab 2: Buffer Pool 2

Instructor: Beom Heyn Kim

beomheynkim@hanyang.ac.kr

Department of Computer Science

Notice

- No late policy for the lab
  - We will call attendance once and call again for those who are absent.
  - If a student is not present at that time, they will be marked absent.

- Do follow the format of submitting the lab file
  - If you don't, you will get one point deduction.

Outline

- LRU Replacement Policy
- Assignment

LRU Replacement Policy

lru_replacer.h

```
class LRUReplacer : public Replacer {
 public:
  /**
   * Create a new LRUReplacer.
   * @param num_pages the maximum number of pages the LRUReplacer will be required to store
   */
  explicit LRUReplacer(size_t num_pages);

  /**
   * Destroys the LRUReplacer.
   */
  ~LRUReplacer() override;

  bool Victim(frame_id_t *frame_id) override;

  void Pin(frame_id_t frame_id) override;

  void Unpin(frame_id_t frame_id) override;

  size_t Size() override;

 private:
  // TODO(student): implement me!
  std::mutex lru_mutex;
  std::list<frame_id_t> unpinned_frames;

};

}  // namespace bustub
```

mutex to protect critical sections

list of unpinned frames

LRU Replacement Policy (Cont.)

lru_replacer.cpp

```
bool LRUReplacer::Victim(frame_id_t *frame_id) {
  lru_mutex.lock();
  if (!unpinned_frames.empty()) {
    *frame_id = unpinned_frames.front();
    unpinned_frames.pop_front();
    lru_mutex.unlock();
    return true;
  }
  lru_mutex.unlock();
  return false;
}
```

Critical Section

Protecting critical sections

We pick the first element in the list of unpinned frames as the victim to evict as it is the oldest that has not been accessed

LRU Replacement Policy (Cont.)

lru_replacer.cpp

```
void LRUReplacer::Pin(frame_id_t frame_id) {
  lru_mutex.lock();
  for (std::list<frame_id_t>::iterator iter = unpinned_frames.begin(); iter != unpinned_frames.end(); iter++) {
    if (*iter == frame_id) {
      unpinned_frames.erase(iter);
      break;
    }
  }
  lru_mutex.unlock();
}
```

Scan the list of unpinned_frames to find the frame to pin. Pinning the frame will remove the frame from the list of unpinned_frames, too.

LRU Replacement Policy (Cont.)

lru_replacer.cpp

```
void LRUReplacer::Unpin(frame_id_t frame_id) {
  lru_mutex.lock();
  bool exist_flag = false;
  for (std::list<frame_id_t>::iterator iter = unpinned_frames.begin(); iter != unpinned_frames.end(); iter++) {
    if (*iter == frame_id) {
      exist_flag = true;
      break;
    }
  }
  if (!exist_flag) {
    unpinned_frames.push_back(frame_id);
  }
  lru_mutex.unlock();
}
```

Scan the list of unpinned_frames to find the frame to pin. Pinning the frame will remove the frame from the list of unpinned_frames, too.

LRU Replacement Policy (Cont.)

lru_replacer.cpp

```cpp
size_t LRUReplacer::Size() {
  return unpinned_frames.size();
}
```

Simply returns the size of unpinned_frames

Don’t forget to test!

```
$ mkdir build
$ cd build
$ make lru_replacer_test
$ ./test/lru_replacer_test
```

Outline

- LRU Replacement Policy
- Assignment

Assignment

- You will need to implement the buffer pool manager. The buffer pool manager's header file and source file are src/include/buffer/buffer_pool_manager.h and src/buffer/buffer_pool_manager.cpp, respectively.
  - Read the source code and comments to prepare for the next lab!

The End