Database Project (SWE3033) (Fall 2024)

Project #1 (100pts, Due date: Oct 8)

Student ID 2018310773

Name 임승재

1. **[5pts]** Write the system setup of your environment. Fill in the blank below.

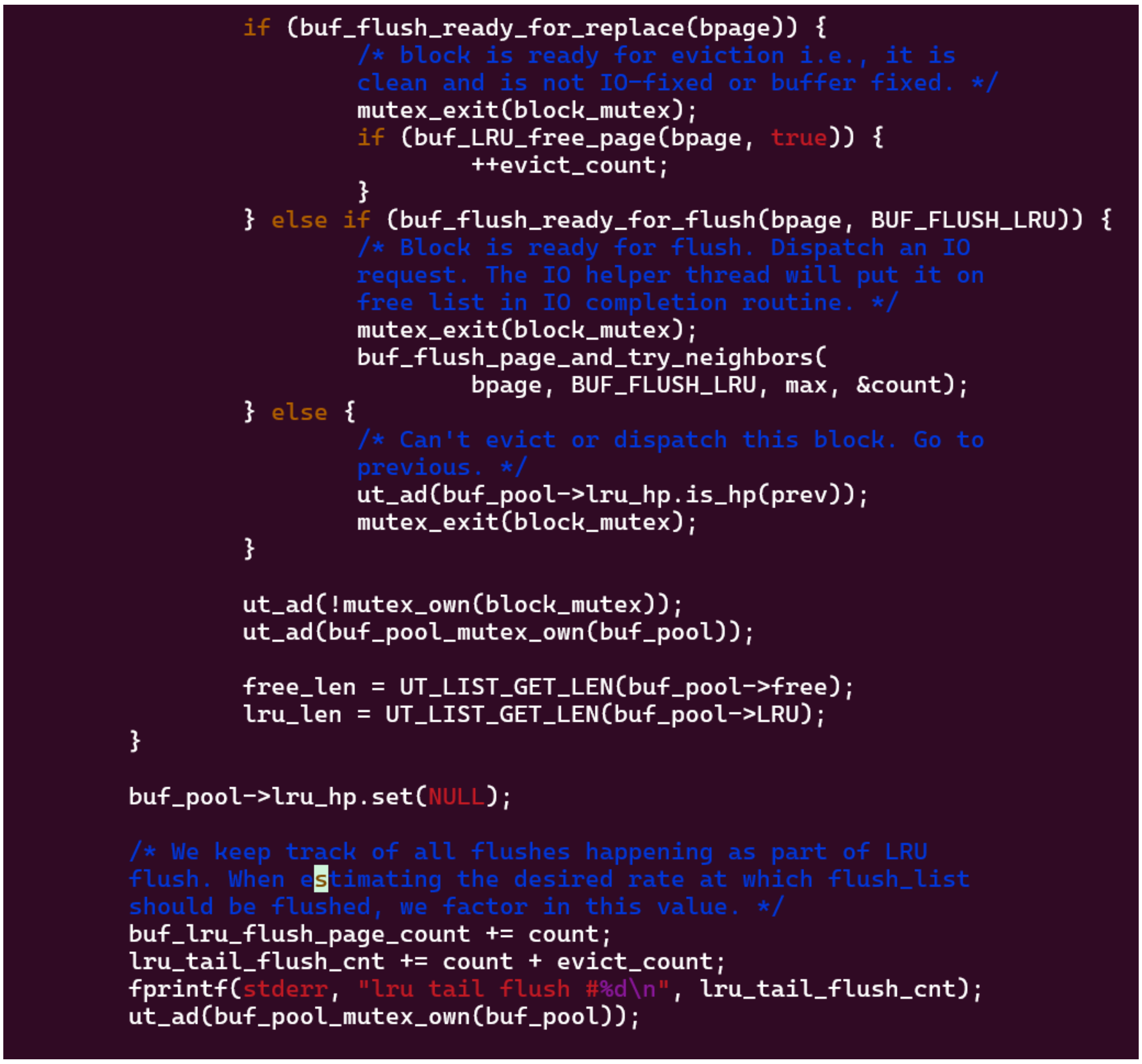
Type	Specification
OS	Windows 10
CPU	Intel Core i7-7700HQ
Memory (RAM)	16GB
Kernel	ubuntu 22.04.3 LTS

2. **[5pts]** Write the benchmark setup of your environment. Fill in the blank below.

Type	Specification
DB Size	1G
Buffer Pool Size	100M, 300M, 500M
Benchmark Tool	tpcc
Runtime	300, 300, 3600(for 100M, 300M, 500M each)
Connections	8

3. **[20pts]** Describe the part of the MySQL source code where you added the code to print the flush types, with a screenshot.

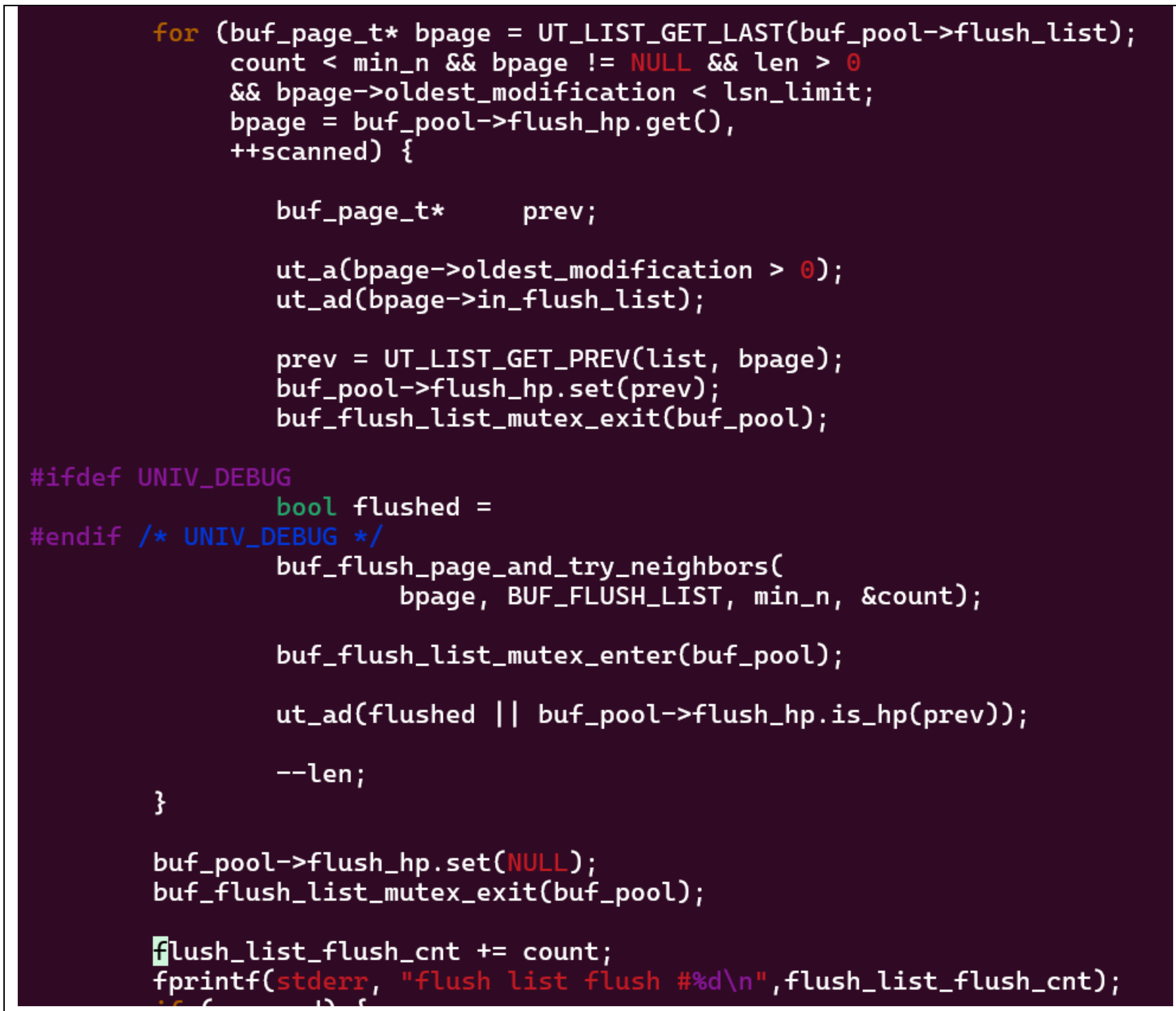
LRU list flush:



```
                if (buf_flush_ready_for_replace(bpage)) {
                        /* block is ready for eviction i.e., it is
                        clean and is not IO-fixed or buffer fixed. */
                        mutex_exit(block_mutex);
                        if (buf_LRU_free_page(bpage, true)) {
                                ++evict_count;
                        }
                } else if (buf_flush_ready_for_flush(bpage, BUF_FLUSH_LRU)) {
                        /* Block is ready for flush. Dispatch an IO
                        request. The IO helper thread will put it on
                        free list in IO completion routine. */
                        mutex_exit(block_mutex);
                        buf_flush_page_and_try_neighbors(
                                bpage, BUF_FLUSH_LRU, max, &count);
                } else {
                        /* Can't evict or dispatch this block. Go to
                        previous. */
                        ut_ad(buf_pool->lru_hp.is_hp(prev));
                        mutex_exit(block_mutex);
                }

                ut_ad(!mutex_own(block_mutex));
                ut_ad(buf_pool_mutex_own(buf_pool));

                free_len = UT_LIST_GET_LEN(buf_pool->free);
                lru_len = UT_LIST_GET_LEN(buf_pool->LRU);
        }

        buf_pool->lru_hp.set(NULL);

        /* We keep track of all flushes happening as part of LRU
        flush. When estimating the desired rate at which flush_list
        should be flushed, we factor in this value. */
        buf_lru_flush_page_count += count;
        lru_tail_flush_cnt += count + evict_count;
        fprintf(stderr, "lru tail flush #%d\n", lru_tail_flush_cnt);
        ut_ad(buf_pool_mutex_own(buf_pool));
```

For the LRU tail flush, I added an fprintf statement to the buf_flush_LRU_list_batch function. When I placed the fprintf statement inside the for loop, the log became too long and wasn't fully output. To resolve this, I used the evict_count and count variables to calculate the total number of LRU tail flushes and printed it outside the for loop.

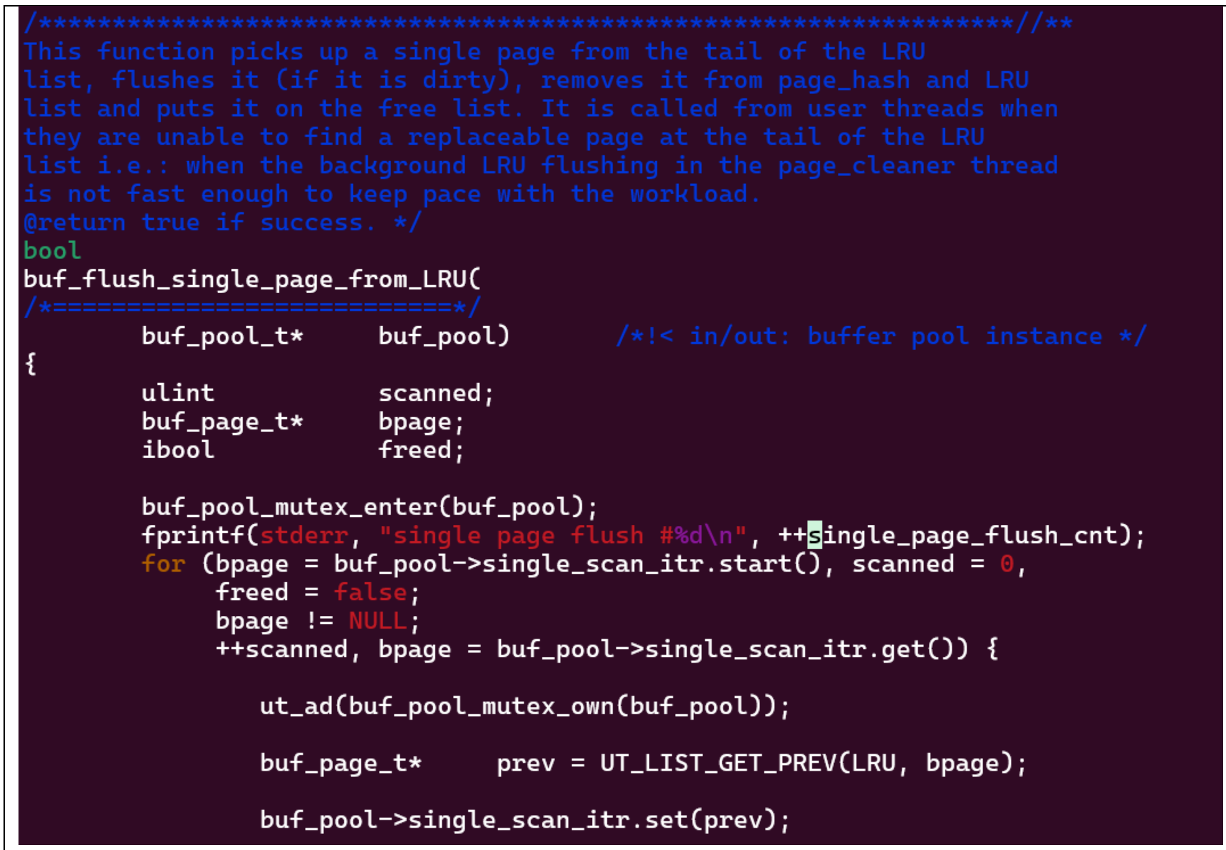
Flush list flush:



```
        for (buf_page_t* bpage = UT_LIST_GET_LAST(buf_pool->flush_list);
             count < min_n && bpage != NULL && len > 0
             && bpage->oldest_modification < lsn_limit;
             bpage = buf_pool->flush_hp.get(),
             ++scanned) {

                buf_page_t*     prev;

                ut_a(bpage->oldest_modification > 0);
                ut_ad(bpage->in_flush_list);

                prev = UT_LIST_GET_PREV(list, bpage);
                buf_pool->flush_hp.set(prev);
                buf_flush_list_mutex_exit(buf_pool);

#ifdef UNIV_DEBUG
                bool flushed =
#endif /* UNIV_DEBUG */
                buf_flush_page_and_try_neighbors(
                        bpage, BUF_FLUSH_LIST, min_n, &count);

                buf_flush_list_mutex_enter(buf_pool);

                ut_ad(flushed || buf_pool->flush_hp.is_hp(prev));

                --len;
        }

        buf_pool->flush_hp.set(NULL);
        buf_flush_list_mutex_exit(buf_pool);

        flush_list_flush_cnt += count;
        fprintf(stderr, "flush list flush #%d\n",flush_list_flush_cnt);
```

For the flush list flush, I added an fprintf statement to the buf_do_flush_list_batch function. When I placed the fprintf statement inside the for loop, the log became too long and wasn't fully output. To resolve this, I used the count variable to calculate the total number of flush list flushes and printed it outside the for loop.

Single Page flush:


```
/******************************************************************//**
This function picks up a single page from the tail of the LRU
list, flushes it (if it is dirty), removes it from page_hash and LRU
list and puts it on the free list. It is called from user threads when
they are unable to find a replaceable page at the tail of the LRU
list i.e.: when the background LRU flushing in the page_cleaner thread
is not fast enough to keep pace with the workload.
@return true if success. */
bool
buf_flush_single_page_from_LRU(
/*===========================*/
	buf_pool_t*	buf_pool)	/*!< in/out: buffer pool instance */
{

	ulint		scanned;
	buf_page_t*	bpage;
	ibool		freed;

	buf_pool_mutex_enter(buf_pool);
	fprintf(stderr, "single page flush #%d\n", ++single_page_flush_cnt);
	for (bpage = buf_pool->single_scan_itr.start(), scanned = 0,
	     freed = false;
	     bpage != NULL;
	     ++scanned, bpage = buf_pool->single_scan_itr.get()) {

		ut_ad(buf_pool_mutex_own(buf_pool));

		buf_page_t*	prev = UT_LIST_GET_PREV(LRU, bpage);

		buf_pool->single_scan_itr.set(prev);
```

For the single page flush, I inserted an `fprintf` statement at the same location as suggested in the assignment PDF. Each time a single page flush occurs, it prints the total number of single page flushes that have happened.

4. **[30pts]** Assume that you have a database with a size of 10 warehouses. Varying the buffer size to 10%, 30%, 50% of database size, calculate the ratio of flush types. You should show the ratio of each flush types and explain about the result.

Buffer Size	10%	30%	50%
LRU List flush (%)	0.6396	87.5628	11.7976
Flush List flush (%)	0.1815	54.3339	45.4846
Single Page flush (%)	0.0105	31.2441	68.7454

Initially, when the buffer size is small, the system relies heavily on LRU Tail Flush to make room for new pages, as there is insufficient space to hold many pages in memory. This results in frequent evictions from the

LRU list. However, as the buffer size increases to 30% and 50%, the reliance on LRU Tail Flush decreases substantially. This is because the larger buffer allows more pages to be retained in memory, reducing the need for frequent evictions.

Additionally, although single page flush occupies a small proportion across all buffer sizes, its ratio is the highest when the buffer size is small. This is because single page flush is a very inefficient operation and does not occur unless absolutely necessary. It happens when the buffer cannot secure enough free pages through asynchronous writes alone.

At the same time, Flush List Flushes increase as the buffer grows. With more memory available, the system can batch multiple pages for flushing in one operation, leading to more Flush List Flushes. This is a more efficient flushing mechanism compared to LRU Tail Flush, as it minimizes disk I/O by writing many pages at once rather than one by one.

5. **[20pts]** It is commonly known that single-page flushes cause performance degradation. Describe the reason for this.

Single-page flushes cause performance degradation primarily because they use synchronous writes, where the system must wait for each write to complete before proceeding. Additionally, flushing one page at a time results in frequent, small I/O operations that increase latency, as each page is written individually. Modern storage systems are optimized for bulk writes, so single-page flushes prevent efficient utilization of I/O bandwidth and system resources. The cumulative effect of these small, synchronous writes leads to slower overall performance compared to other flush mechanisms.

6. **[20pts]** Explain the checkpoint, and why is it necessary? Also, what happens when the checkpoint occurs in the MySQL? Explain with the three main lists in the MySQL.

A checkpoint in MySQL is the process of writing modified pages from memory to disk to ensure data consistency and support crash recovery. It helps reduce the amount of work needed to recover the system after a crash and optimizes performance by managing when data is written to disk. During a checkpoint, MySQL selects pages from the flush list and writes them to disk. These pages are not freed, but marked as clean pages in LRU list. Also, the redo log is updated to mark which log entries have been flushed to disk. This ensures that, in the event of a crash, MySQL can start recovery from the last checkpoint instead of replaying the entire redo log.