**Project Structure**

**The source code is organized as follows:**

**Copy code**

**assign2/**

**├── README.md**

**├── Makefile**

**├── buffer\_mgr.h**

**├── buffer\_mgr.c**

**├── buffer\_mgr\_stat.c**

**├── buffer\_mgr\_stat.h**

**├── dberror.c**

**├── dberror.h**

**├── dt.h**

**├── storage\_mgr.h**

**├── test\_assign2\_1.c**

**└── test\_helper.h**

**Key Data Structures**

**BM\_BufferPool**

**cCopy code**

**typedef struct BM\_BufferPool {**

**char \*pageFile;**

**int numPages;**

**ReplacementStrategy strategy;**

**void \*mgmtData; // bookkeeping info**

**} BM\_BufferPool;**

**BM\_PageHandle**

**cCopy code**

**typedef struct BM\_PageHandle {**

**PageNumber pageNum;**

**char \*data;**

**} BM\_PageHandle;**

**Buffer Pool Functions**

**Initialization and Shutdown**

* **initBufferPool: Initializes a new buffer pool.**
* **shutdownBufferPool: Frees resources associated with the buffer pool.**
* **forceFlushPool: Writes all dirty pages back to disk.**

**Page Management Functions**

* **pinPage: Pins a page into the buffer pool.**
* **unpinPage: Unpins a page from the buffer pool.**
* **markDirty: Marks a page as dirty.**
* **forcePage: Forces a page to be written back to disk.**

**Statistics Functions**

* **getFrameContents: Returns an array of page numbers in the frames.**
* **getDirtyFlags: Returns an array indicating if frames are dirty.**
* **getFixCounts: Returns the fix counts for each frame.**
* **getNumReadIO: Returns the number of read I/O operations.**
* **getNumWriteIO: Returns the number of write I/O operations.**

**Replacement Strategies**

**FIFO (First-In-First-Out)**

**In this strategy, the oldest page in the buffer pool is replaced first when a new page needs to be loaded.**

**LRU (Least Recently Used)**

**In this strategy, the least recently used page is replaced when a new page needs to be loaded.**

**Error Handling**

**The implementation includes error handling through defined return codes in dberror.h. Each function checks for errors and returns appropriate codes.**

**Example Usage**

**Here’s how you might use the buffer manager in your application:**

**cCopy code**

**BM\_BufferPool \*bm = MAKE\_POOL();**

**initBufferPool(bm, "example.dat", 3, RS\_FIFO, NULL);**

**BM\_PageHandle \*pageHandle = MAKE\_PAGE\_HANDLE();**

**pinPage(bm, pageHandle, 0); // Pin page 0**

**// Work with the page...**

**markDirty(bm, pageHandle); // Mark as dirty**

**unpinPage(bm, pageHandle); // Unpin the page**

**shutdownBufferPool(bm);**

**Testing**

**The test\_assign2\_1.c file contains various test cases to ensure the buffer manager functions correctly under both FIFO and LRU strategies. You can extend these tests to cover additional scenarios.**

**Build Instructions**

**To compile the project, navigate to the assign2 directory and run:**

**bashCopy code**

**make**

**This will generate the necessary binaries for testing.**

**Conclusion**

**This buffer manager provides a foundational structure for managing memory pages efficiently. Future extensions could include implementing additional replacement strategies or making the buffer manager thread-safe.**