BLINK DB

1.0

Generated by Doxygen 1.13.2

1 BLINK DB Documentation 1

1.1 Overview . . . 1
1.2 Architecture . . . 1
1.2.1 Storage Engine (Part A) . . . 1
1.2.2 Advanced Storage Engine (Part B) . . . 2
1.2.3 Network Server (Part B) . . . 2
1.3 Performance Characteristics . . . 2
1.3.1 Benchmark Results . . . 2
1.3.2 Key Features . . . 2
1.4 Getting Started . . . 3
1.4.1 Building the Project . . . 3
1.4.2 Running the Server . . . 3
1.4.3 Running the Client . . . 3
1.4.4 Running Benchmarks . . . 3
1.4.5 Running Part A REPL . . . 3
1.5 Implementation Differences . . . 3
1.5.1 Part A vs Part B Storage Engines . . . 3
1.5.2 Key Classes in Part B . . . 3
1.6 Project Structure . . . 4
1.7 API Documentation . . . 4
1.7.1 Basic Storage Engine API (Part A) . . . 4
1.7.2 Advanced Storage Engine API (Part B) . . . 5
1.7.3 LRUCache API (Part B) . . . 5
1.7.4 Network Server API (Part B) . . . 5
1.7.5 Network Client API (Part B) . . . 6
1.8 Performance Optimization . . . 6
1.8.1 Memory Management (Part B) . . . 6
1.8.2 Disk Operations . . . 6
1.8.3 Network Optimization (Part B) . . . 6
1.9 Future Improvements . . . 6
1.9.1 Performance . . . 6
1.9.2 Features . . . 7
1.9.3 Reliability . . . 7

2 Class Index 9

2.1 Class List . . . 9

3 File Index 11

3.1 File List . . . 11

4 Class Documentation 13

4.1 StorageEngine::BatchEntry Struct Reference . . . 13
4.1.1 Member Data Documentation . . . 13

4.1.1.1 key . . . 13
4.1.1.2 value . . . 13
4.2 StorageEngine::DiskEntry Struct Reference . . . 13
4.2.1 Member Data Documentation . . . 14
4.2.1.1 offset . . . 14
4.2.1.2 size . . . 14
4.3 DiskStorage Class Reference . . . 14
4.3.1 Constructor & Destructor Documentation . . . 15
4.3.1.1 DiskStorage() . . . 15
4.3.1.2 ~DiskStorage() . . . 15
4.3.2 Member Function Documentation . . . 15
4.3.2.1 ensure_directory_exists() . . . 15
4.3.2.2 get() . . . 16
4.3.2.3 get_executable_path() . . . 16
4.3.2.4 load_data() . . . 16
4.3.2.5 put() . . . 17
4.3.2.6 remove() . . . 17
4.3.2.7 save_data() . . . 17
4.3.3 Member Data Documentation . . . 18
4.3.3.1 data_ . . . 18
4.3.3.2 data_file_ . . . 18
4.3.3.3 mutex_ . . . 18
4.4 LRUCache Class Reference . . . 18
4.4.1 Constructor & Destructor Documentation . . . 19
4.4.1.1 LRUCache() . . . 19
4.4.1.2 ~LRUCache() . . . 19
4.4.2 Member Function Documentation . . . 20
4.4.2.1 capacity() . . . 20
4.4.2.2 evict_lru() . . . 20
4.4.2.3 get() . . . 20
4.4.2.4 move_to_front() . . . 21
4.4.2.5 put() . . . 21
4.4.2.6 remove() . . . 21
4.4.2.7 size() . . . 21
4.4.3 Member Data Documentation . . . 22
4.4.3.1 cache_ . . . 22
4.4.3.2 capacity_ . . . 22
4.4.3.3 head_ . . . 22
4.4.3.4 mutex_ . . . 22
4.4.3.5 tail_ . . . 22
4.5 NetworkClient Class Reference . . . 22
4.5.1 Constructor & Destructor Documentation . . . 23

4.5.1.1 NetworkClient() . . . 23
4.5.1.2 ~NetworkClient() . . . 23
4.5.2 Member Function Documentation . . . 23
4.5.2.1 parse_response() . . . 23
4.5.2.2 send_command() . . . 23
4.5.3 Member Data Documentation . . . 24
4.5.3.1 server_addr . . . 24
4.5.3.2 sock . . . 24
4.6 LRUCache::Node Struct Reference . . . 24
4.6.1 Constructor & Destructor Documentation . . . 25
4.6.1.1 Node() . . . 25
4.6.2 Member Data Documentation . . . 25
4.6.2.1 key . . . 25
4.6.2.2 next . . . 25
4.6.2.3 prev . . . 25
4.6.2.4 value . . . 25
4.7 Server Class Reference . . . 25
4.7.1 Constructor & Destructor Documentation . . . 26
4.7.1.1 Server() . . . 26
4.7.1.2 ~Server() . . . 27
4.7.2 Member Function Documentation . . . 27
4.7.2.1 encode_resp() . . . 27
4.7.2.2 handle_client_data() . . . 27
4.7.2.3 handle_new_connection() . . . 28
4.7.2.4 process_command() . . . 28
4.7.2.5 run() . . . 29
4.7.2.6 set_nonblocking() . . . 29
4.7.2.7 setup_kqueue() . . . 29
4.7.2.8 setup_server() . . . 30
4.7.2.9 stop() . . . 30
4.7.3 Member Data Documentation . . . 30
4.7.3.1 client_buffers . . . 30
4.7.3.2 kq . . . 31
4.7.3.3 LISTEN_BACKLOG . . . 31
4.7.3.4 MAX_EVENTS . . . 31
4.7.3.5 PORT . . . 31
4.7.3.6 server_fd . . . 31
4.7.3.7 should_stop . . . 31
4.7.3.8 storage . . . 31
4.8 StorageEngine Class Reference . . . 31
4.8.1 Constructor & Destructor Documentation . . . 33
4.8.1.1 StorageEngine() [1/2] . . . 33

4.8.1.2 ~StorageEngine() [1/2] . . . 33
4.8.1.3 StorageEngine() [2/2] . . . 33
4.8.1.4 ~StorageEngine() [2/2] . . . 34
4.8.2 Member Function Documentation . . . 34
4.8.2.1 async_write_worker() . . . 34
4.8.2.2 clear() [1/2] . . . 34
4.8.2.3 clear() [2/2] . . . 34
4.8.2.4 del() [1/2] . . . 35
4.8.2.5 del() [2/2] . . . 35
4.8.2.6 flush_write_buffer() . . . 35
4.8.2.7 force_flush() [1/2] . . . 36
4.8.2.8 force_flush() [2/2] . . . 36
4.8.2.9 get() [1/3] . . . 37
4.8.2.10 get() [2/3] . . . 37
4.8.2.11 get() [3/3] . . . 37
4.8.2.12 load_disk_index() . . . 38
4.8.2.13 pending_write_count() . . . 38
4.8.2.14 put() . . . 38
4.8.2.15 remove_from_disk_index() . . . 39
4.8.2.16 save_disk_index() . . . 39
4.8.2.17 set() [1/2] . . . 40
4.8.2.18 set() [2/2] . . . 40
4.8.2.19 size() [1/2] . . . 41
4.8.2.20 size() [2/2] . . . 41
4.8.2.21 stop_async_writer() . . . 41
4.8.2.22 sync() . . . 41
4.8.2.23 update_disk_index() . . . 41
4.8.3 Member Data Documentation . . . 42
4.8.3.1 access_order . . . 42
4.8.3.2 BATCH_SIZE . . . 42
4.8.3.3 cache_ . . . 42
4.8.3.4 data_ . . . 42
4.8.3.5 DATA_FILE . . . 42
4.8.3.6 DISK_DIR . . . 43
4.8.3.7 disk_index . . . 43
4.8.3.8 disk_storage_ . . . 43
4.8.3.9 INDEX_FILE . . . 43
4.8.3.10 MAX_CACHE_SIZE . . . 43
4.8.3.11 MAX_KEY_SIZE . . . 43
4.8.3.12 MAX_VALUE_SIZE . . . 43
4.8.3.13 mutex_ . . . 43
4.8.3.14 pending_writes . . . 43

4.8.3.15 running_ . . . 43
4.8.3.16 write_buffer . . . 44
4.8.3.17 write_cv_ . . . 44
4.8.3.18 write_mutex_ . . . 44
4.8.3.19 write_queue_ . . . 44
4.8.3.20 write_thread_ . . . 44

5 File Documentation **45**

5.1 docs/mainpage.md File Reference . . . 45
5.2 part-a/src/repl.cpp File Reference . . . 45
5.2.1 Function Documentation . . . 46
5.2.1.1 main() . . . 46
5.2.1.2 printUsage() . . . 46
5.3 part-a/src/storage_engine.cpp File Reference . . . 46
5.4 part-b/src/storage_engine.cpp File Reference . . . 47
5.5 part-a/src/storage_engine.h File Reference . . . 47
5.6 storage_engine.h . . . 48
5.7 part-b/src/storage_engine.h File Reference . . . 49
5.8 storage_engine.h . . . 50
5.9 part-b/src/main_server.cpp File Reference . . . 55
5.9.1 Function Documentation . . . 55
5.9.1.1 main() . . . 55
5.9.1.2 signal_handler() . . . 56
5.9.2 Variable Documentation . . . 56
5.9.2.1 g_server . . . 56
5.10 part-b/src/network_client.cpp File Reference . . . 56
5.10.1 Function Documentation . . . 57
5.10.1.1 main() . . . 57
5.11 part-b/src/network_server.cpp File Reference . . . 57
5.11.1 Macro Definition Documentation . . . 57
5.11.1.1 TCP_NODELAY . . . 57
5.12 part-b/src/network_server.h File Reference . . . 58
5.13 network_server.h . . . 58

Index **61**

Chapter 1

BLINK DB Documentation

1.1 Overview

BLINK DB is a high-performance key-value store implementation with both in-memory and disk-based storage capabilities. The project is divided into two main parts:

1. **Part A: Basic Storage Engine**
    - Simple in-memory storage with access order tracking
    - Basic disk-based persistence with binary format
    - Synchronous write operations
    - Simple disk index management
    - REPL (Read-Eval-Print Loop) interface
2. **Part B: Advanced Network Server**
    - Redis protocol (RESP) support
    - Non-blocking I/O using kqueue/epoll
    - High-concurrency client handling
    - Advanced LRU cache implementation
    - Asynchronous disk operations
    - Network client implementation

1.2 Architecture

1.2.1 Storage Engine (Part A)

The basic storage engine provides core functionality for storing and retrieving key-value pairs. It features:

- Simple in-memory hash map storage
- Access order tracking for basic LRU behavior
- Binary format disk persistence
- Synchronous write operations
- Simple disk index management
- REPL interface for interactive testing

1.2.2 Advanced Storage Engine (Part B)

The advanced storage engine includes sophisticated caching and async operations:

- **LRUCache class**: Thread-safe LRU cache with doubly-linked list implementation
- **DiskStorage class**: Persistent storage with automatic directory management
- **StorageEngine class**: Main engine with async write worker thread
- Write buffering and batch operations
- Cross-platform executable path detection

1.2.3 Network Server (Part B)

The network server implements the Redis protocol and provides:

- Non-blocking I/O using kqueue/epoll
- High-concurrency client handling
- RESP protocol implementation
- Signal handling for graceful shutdown
- Network client implementation for testing

1.3 Performance Characteristics

1.3.1 Benchmark Results

1. **Low Load (1000 requests, 10 connections)**
   - SET: 66,666 ops/sec
   - GET: 83,333 ops/sec
2. **High Load (100,000 requests, 100 connections)**
   - SET: 167,785 ops/sec
   - GET: 168,067 ops/sec

1.3.2 Key Features

- High throughput under load
- Low latency for most operations
- Good scaling with concurrency
- Efficient resource utilization

1.4 Getting Started

1.4.1 Building the Project

```
# Part A: Storage Engine
cd part-a
make clean
make

# Part B: Network Server
cd part-b
make clean
make
```

1.4.2 Running the Server

```
cd part-b
./blinkdb_server
```

1.4.3 Running the Client

```
cd part-b
./blinkdb_client
```

1.4.4 Running Benchmarks

```
# Using redis-benchmark
redis-benchmark -p 9001 -n 100000 -c 100

# Using built-in benchmark
./benchmark 100000 100
```

1.4.5 Running Part A REPL

```
cd part-a
./blinkdb
```

1.5 Implementation Differences

1.5.1 Part A vs Part B Storage Engines

Feature	Part A (Basic)	Part B (Advanced)
Storage Format	Binary format (data.dat, index.dat)	Text format (data.txt)
Caching	Simple access order tracking	Thread-safe LRU cache with doubly-linked list
Write Operations	Synchronous, immediate flush	Asynchronous with background worker thread
Thread Safety	Basic mutex protection	Advanced thread-safe design with condition variables
Disk Management	Manual directory creation	Automatic executable path detection
Memory Management	Simple hash map	Sophisticated cache with eviction policies
Interface	REPL command-line	Network server with RESP protocol

1.5.2 Key Classes in Part B

- **LRUCache**: Implements thread-safe LRU eviction with O(1) operations

- **DiskStorage**: Handles persistent storage with automatic path management
- **StorageEngine**: Main engine coordinating cache and disk operations
- **Server**: Network server with kqueue/epoll for high concurrency
- **NetworkClient**: Client implementation for testing and benchmarking

1.6 Project Structure

```
.
+-- part-a/                # Basic Storage Engine
|   +-- src/
|   |   +-- storage_engine.cpp    # Basic storage implementation
|   |   +-- storage_engine.h      # Basic storage header
|   |   +-- repl.cpp             # REPL interface
|   +-- disk_storage/            # Disk storage files
|   |   +-- data.dat
|   |   +-- index.dat
|   +-- blinkdb                  # Compiled executable
+-- part-b/                # Advanced Network Server
|   +-- src/
|   |   +-- main_server.cpp      # Server main entry point
|   |   +-- network_server.cpp   # Network server implementation
|   |   +-- network_server.h     # Network server header
|   |   +-- network_client.cpp   # Network client implementation
|   |   +-- storage_engine.cpp   # Advanced storage implementation
|   |   +-- storage_engine.h     # Advanced storage header
|   +-- benchmark.cpp            # Performance benchmark tool
|   +-- disk_storage/            # Disk storage files
|   |   +-- data.txt
|   +-- blinkdb_server           # Compiled server executable
|   +-- blinkdb_client           # Compiled client executable
|   +-- benchmark                # Compiled benchmark executable
+-- docs/                # Documentation
    +-- mainpage.md
    +-- doxygen/           # Generated documentation
        +-- html/
        +-- latex/
```

1.7 API Documentation

1.7.1 Basic Storage Engine API (Part A)

The basic storage engine provides the following main interfaces:

- `set(key, value)`: Store a key-value pair
- `get(key)`: Retrieve a value by key
- `del(key)`: Delete a key-value pair
- `clear()`: Remove all key-value pairs
- `force_flush()`: Force flush write buffer to disk
- `size()`: Get total number of entries

1.7.2 Advanced Storage Engine API (Part B)

The advanced storage engine provides enhanced functionality:

- `set(key, value)`: Store a key-value pair (alias for put)
- `put(key, value)`: Store a key-value pair with async write
- `get(key)`: Retrieve a value by key
- `get(key, value)`: Retrieve a value by key (reference version)
- `del(key)`: Delete a key-value pair
- `clear()`: Remove all key-value pairs
- `force_flush()`: Force flush pending writes
- `sync()`: Synchronize all pending operations
- `size()`: Get cache size
- `pending_write_count()`: Get number of pending writes
- `stop_async_writer()`: Stop the async write worker

1.7.3 LRUCache API (Part B)

The LRU cache provides thread-safe caching:

- `get(key, value)`: Retrieve value from cache
- `put(key, value)`: Store value in cache
- `remove(key)`: Remove key from cache
- `capacity()`: Get cache capacity
- `size()`: Get current cache size

1.7.4 Network Server API (Part B)

The network server implements:

- RESP protocol parsing and encoding
- Client connection management with kqueue/epoll
- Command processing (SET, GET, DEL, PING, etc.)
- Response formatting
- Signal handling for graceful shutdown

1.7.5 Network Client API (Part B)

The network client provides:

- TCP connection management
- RESP protocol communication
- Command sending and response parsing
- Error handling and connection testing

1.8 Performance Optimization

1.8.1 Memory Management (Part B)

- **LRU Cache**: Thread-safe doubly-linked list implementation for efficient cache management
- **Write Buffering**: Batch operations for better disk I/O performance
- **Memory-efficient Data Structures**: Optimized hash maps and linked lists
- **Async Write Worker**: Background thread for non-blocking disk operations

1.8.2 Disk Operations

- **Part A**: Synchronous binary format with simple index management
- **Part B**: Asynchronous writes with background worker thread
- **Cross-platform Path Detection**: Automatic executable path detection for storage location
- **Batch Index Updates**: Efficient disk index management

1.8.3 Network Optimization (Part B)

- **Non-blocking I/O**: kqueue/epoll for high-concurrency handling
- **RESP Protocol**: Efficient Redis-compatible protocol implementation
- **Signal Handling**: Graceful shutdown with SIGINT/SIGTERM support
- **Client Connection Management**: Efficient client buffer management

1.9 Future Improvements

1.9.1 Performance

- Implement connection pooling
- Add read-ahead caching
- Optimize disk index structure

1.9.2 Features

- Add transaction support
- Implement pub/sub
- Add data compression

1.9.3 Reliability

- Add data replication
- Implement checkpointing
- Add data validation

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

StorageEngine::BatchEntry 13
StorageEngine::DiskEntry 13
DiskStorage 14
LRUCache 18
NetworkClient 22
LRUCache::Node 24
Server 25
StorageEngine 31

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

part-a/src/repl.cpp 45
part-a/src/storage_engine.cpp 46
part-a/src/storage_engine.h 47
part-b/src/main_server.cpp 55
part-b/src/network_client.cpp 56
part-b/src/network_server.cpp 57
part-b/src/network_server.h 58
part-b/src/storage_engine.cpp 47
part-b/src/storage_engine.h 49

Chapter 4

Class Documentation

4.1 StorageEngine::BatchEntry Struct Reference

Public Attributes

- std::string key
- std::string value

4.1.1 Member Data Documentation

4.1.1.1 key

```
std::string StorageEngine::BatchEntry::key
```

4.1.1.2 value

```
std::string StorageEngine::BatchEntry::value
```

The documentation for this struct was generated from the following file:

- part-a/src/storage_engine.h

4.2 StorageEngine::DiskEntry Struct Reference

Public Attributes

- size_t offset
- size_t size

4.2.1 Member Data Documentation

4.2.1.1 offset

```
size_t StorageEngine::DiskEntry::offset
```

4.2.1.2 size

```
size_t StorageEngine::DiskEntry::size
```

The documentation for this struct was generated from the following file:

- part-a/src/storage_engine.h

4.3 DiskStorage Class Reference

```
#include <storage_engine.h>
```

Public Member Functions

- DiskStorage ()
- ~DiskStorage ()
- bool get (const std::string &key, std::string &value)
- bool put (const std::string &key, const std::string &value)
- void remove (const std::string &key)

Private Member Functions

- std::filesystem::path get_executable_path ()
- void ensure_directory_exists ()
- void load_data ()
- void save_data ()

Private Attributes

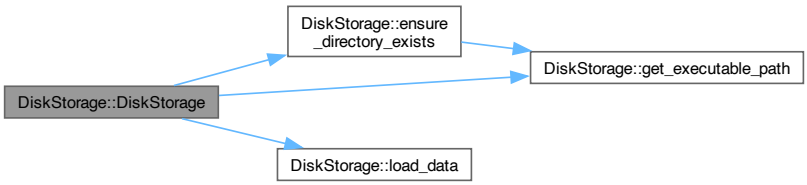
- std::string data_file_
- std::mutex mutex_
- std::unordered_map< std::string, std::string > data_

4.3.1 Constructor & Destructor Documentation

4.3.1.1 DiskStorage()

```
DiskStorage::DiskStorage ()  [inline]
```

Here is the call graph for this function:



4.3.1.2 ~DiskStorage()

```
DiskStorage::~DiskStorage ()  [inline]
```

Here is the call graph for this function:

4.3.2 Member Function Documentation

4.3.2.1 ensure_directory_exists()

```
void DiskStorage::ensure_directory_exists ()  [inline], [private]
```

Here is the call graph for this function:

Here is the caller graph for this function:

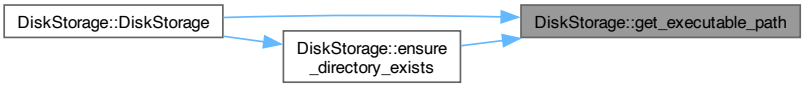
4.3.2.2 get()

```
bool DiskStorage::get (
            const std::string & key,
            std::string & value)  [inline]
```

4.3.2.3 get_executable_path()

```
std::filesystem::path DiskStorage::get_executable_path ()  [inline], [private]
```

Here is the caller graph for this function:



4.3.2.4 load_data()

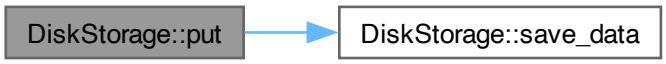
```
void DiskStorage::load_data ()  [inline], [private]
```

Here is the caller graph for this function:

4.3.2.5 put()

```
bool DiskStorage::put (
            const std::string & key,
            const std::string & value)  [inline]
```

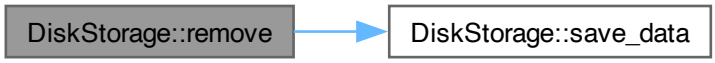
Here is the call graph for this function:



4.3.2.6 remove()

```
void DiskStorage::remove (
            const std::string & key)  [inline]
```

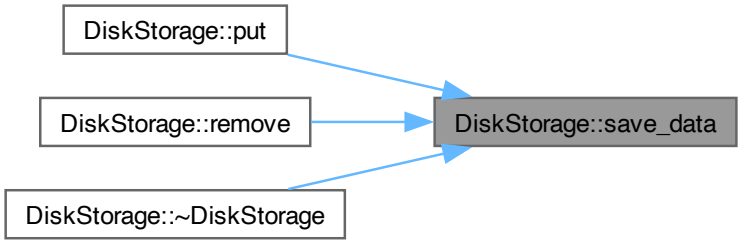
Here is the call graph for this function:



4.3.2.7 save_data()

```
void DiskStorage::save_data ()  [inline], [private]
```

Here is the caller graph for this function:



4.3.3 Member Data Documentation

4.3.3.1 data_

```
std::unordered_map<std::string, std::string> DiskStorage::data_  [private]
```

4.3.3.2 data_file_

```
std::string DiskStorage::data_file_  [private]
```

4.3.3.3 mutex_

```
std::mutex DiskStorage::mutex_  [private]
```

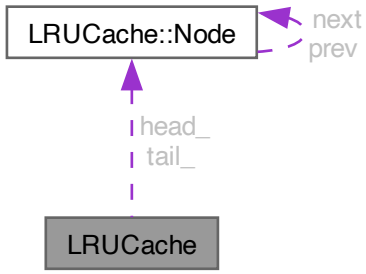
The documentation for this class was generated from the following file:

- part-b/src/storage_engine.h

4.4 LRUCache Class Reference

```
#include <storage_engine.h>
```

Collaboration diagram for LRUCache:



Classes

- struct Node

Public Member Functions

- LRUCache (size_t capacity)
- ~LRUCache ()
- size_t capacity () const
- size_t size () const
- bool get (const std::string &key, std::string &value)
- bool put (const std::string &key, const std::string &value)
- bool remove (const std::string &key)

Private Member Functions

- void move_to_front (Node *node)
- void evict_lru ()

Private Attributes

- size_t capacity_
- std::unordered_map< std::string, Node * > cache_
- Node * head_
- Node * tail_
- std::mutex mutex_

4.4.1 Constructor & Destructor Documentation

4.4.1.1 LRUCache()

```
LRUCache::LRUCache (
            size_t capacity)  [inline]
```

Here is the call graph for this function:

4.4.1.2 ~LRUCache()

```
LRUCache::~LRUCache ()  [inline]
```

4.4.2 Member Function Documentation

4.4.2.1 capacity()

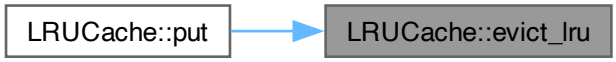
```
size_t LRUCache::capacity () const  [inline]
```

Here is the caller graph for this function:

4.4.2.2 evict_lru()

```
void LRUCache::evict_lru ()  [inline], [private]
```

Here is the caller graph for this function:



4.4.2.3 get()

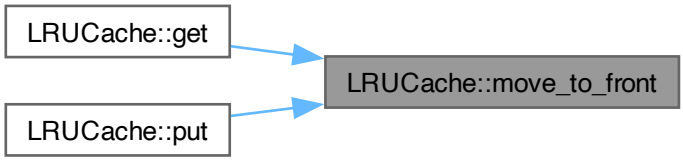
```
bool LRUCache::get (
            const std::string & key,
            std::string & value)  [inline]
```

Here is the call graph for this function:

4.4.2.4 move_to_front()

```
void LRUCache::move_to_front (
            Node * node)  [inline], [private]
```

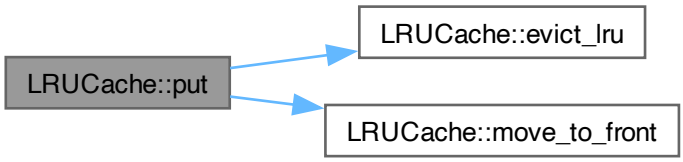
Here is the caller graph for this function:



4.4.2.5 put()

```
bool LRUCache::put (
            const std::string & key,
            const std::string & value)  [inline]
```

Here is the call graph for this function:



4.4.2.6 remove()

```
bool LRUCache::remove (
            const std::string & key)  [inline]
```

4.4.2.7 size()

```
size_t LRUCache::size () const  [inline]
```

4.4.3 Member Data Documentation

4.4.3.1 cache_

```
std::unordered_map<std::string, Node*> LRUCache::cache_  [private]
```

4.4.3.2 capacity_

```
size_t LRUCache::capacity_  [private]
```

4.4.3.3 head_

```
Node* LRUCache::head_  [private]
```

4.4.3.4 mutex_

```
std::mutex LRUCache::mutex_  [private]
```

4.4.3.5 tail_

```
Node* LRUCache::tail_  [private]
```

The documentation for this class was generated from the following file:

- part-b/src/storage_engine.h

4.5 NetworkClient Class Reference

Public Member Functions

- NetworkClient (const char *host="127.0.0.1", int port=9001)
- ~NetworkClient ()
- std::string send_command (const std::string &command)

Private Member Functions

- std::string parse_response (const std::string &resp)

Private Attributes

- int sock
- struct sockaddr_in server_addr

4.5.1 Constructor & Destructor Documentation

4.5.1.1 NetworkClient()

```
NetworkClient::NetworkClient (
            const char * host = "127.0.0.1",
            int port = 9001)  [inline]
```

4.5.1.2 ~NetworkClient()

```
NetworkClient::~NetworkClient ()  [inline]
```

4.5.2 Member Function Documentation

4.5.2.1 parse_response()

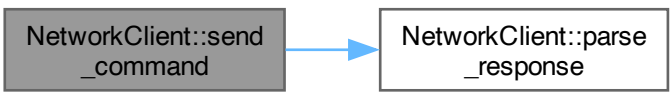
```
std::string NetworkClient::parse_response (
            const std::string & resp)  [inline], [private]
```

Here is the caller graph for this function:

4.5.2.2 send_command()

```
std::string NetworkClient::send_command (
            const std::string & command)  [inline]
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.5.3 Member Data Documentation

4.5.3.1 server_addr

```
struct sockaddr_in NetworkClient::server_addr  [private]
```

4.5.3.2 sock

```
int NetworkClient::sock  [private]
```

The documentation for this class was generated from the following file:

- part-b/src/network_client.cpp

4.6 LRUCache::Node Struct Reference

Collaboration diagram for LRUCache::Node:

Public Member Functions

- Node (const std::string &k, const std::string &v)

Public Attributes

- std::string key
- std::string value
- Node * prev
- Node * next

4.6.1 Constructor & Destructor Documentation

4.6.1.1 Node()

```
LRUCache::Node::Node (
            const std::string & k,
            const std::string & v)  [inline]
```

4.6.2 Member Data Documentation

4.6.2.1 key

```
std::string LRUCache::Node::key
```

4.6.2.2 next

```
Node* LRUCache::Node::next
```

4.6.2.3 prev

```
Node* LRUCache::Node::prev
```

4.6.2.4 value

```
std::string LRUCache::Node::value
```

The documentation for this struct was generated from the following file:

- part-b/src/storage_engine.h

4.7 Server Class Reference

```
#include <network_server.h>
```

Public Member Functions

- Server ()
- ~Server ()
- void run ()
- void stop ()

Private Member Functions

- void setup_server ()
- void setup_kqueue ()
- void set_nonblocking (int fd)
- void handle_new_connection ()
- void handle_client_data (int client_fd)
- std::string process_command (const std::string &command)
- std::string encode_resp (const std::string &response)

Private Attributes

- int server_fd = -1
- int kq = -1
- bool should_stop = false
- std::unique_ptr< StorageEngine > storage
- std::unordered_map< int, std::string > client_buffers

Static Private Attributes

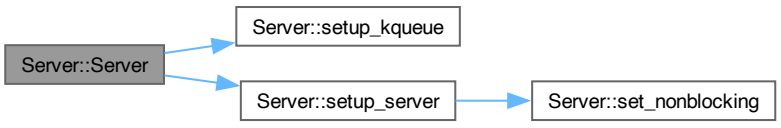
- static constexpr int PORT = 9001
- static constexpr int LISTEN_BACKLOG = 128
- static constexpr int MAX_EVENTS = 1024

4.7.1 Constructor & Destructor Documentation

4.7.1.1 Server()

```
Server::Server ()
```

Here is the call graph for this function:



4.7.1.2 ~Server()

```
Server::~Server ()
```

Here is the call graph for this function:

4.7.2 Member Function Documentation

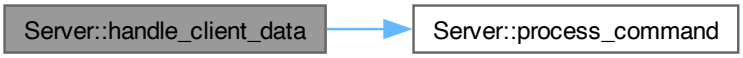
4.7.2.1 encode_resp()

```
std::string Server::encode_resp (
            const std::string & response)  [private]
```

4.7.2.2 handle_client_data()

```
void Server::handle_client_data (
            int client_fd)  [private]
```

Here is the call graph for this function:

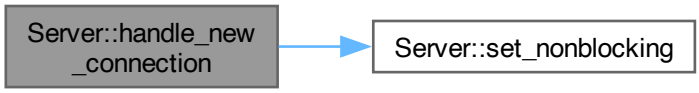


Here is the caller graph for this function:

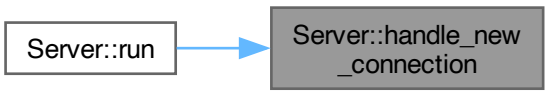
4.7.2.3 handle_new_connection()

```
void Server::handle_new_connection ()  [private]
```

Here is the call graph for this function:



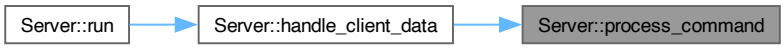
Here is the caller graph for this function:



4.7.2.4 process_command()

```
std::string Server::process_command (
            const std::string & command)  [private]
```

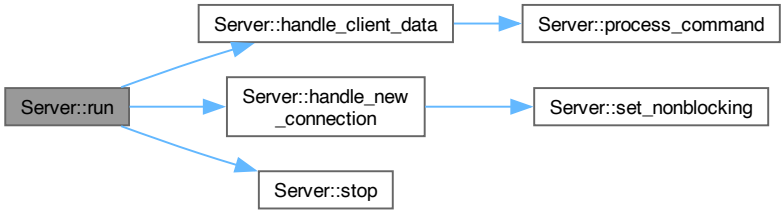
Here is the caller graph for this function:



4.7.2.5 run()

```
void Server::run ()
```

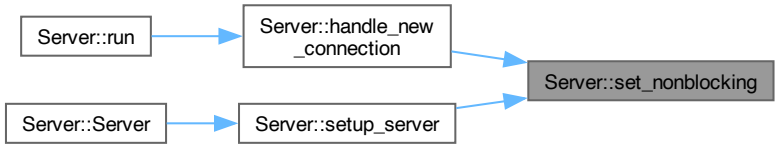
Here is the call graph for this function:



4.7.2.6 set_nonblocking()

```
void Server::set_nonblocking (
            int fd)  [private]
```

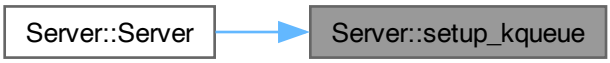
Here is the caller graph for this function:



4.7.2.7 setup_kqueue()

```
void Server::setup_kqueue ()  [private]
```

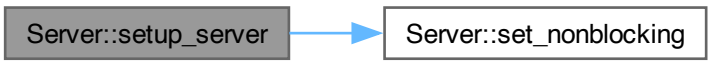
Here is the caller graph for this function:



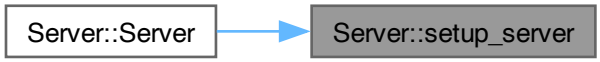
4.7.2.8 setup_server()

```
void Server::setup_server ()  [private]
```

Here is the call graph for this function:



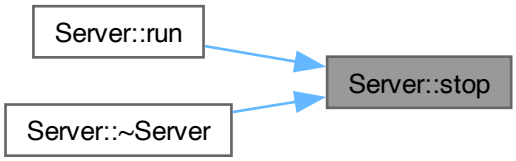
Here is the caller graph for this function:



4.7.2.9 stop()

```
void Server::stop ()
```

Here is the caller graph for this function:



4.7.3 Member Data Documentation

4.7.3.1 client_buffers

```
std::unordered_map<int, std::string> Server::client_buffers  [private]
```

4.7.3.2 kq

```
int Server::kq = -1  [private]
```

4.7.3.3 LISTEN_BACKLOG

```
int Server::LISTEN_BACKLOG = 128  [static], [constexpr], [private]
```

4.7.3.4 MAX_EVENTS

```
int Server::MAX_EVENTS = 1024  [static], [constexpr], [private]
```

4.7.3.5 PORT

```
int Server::PORT = 9001  [static], [constexpr], [private]
```

4.7.3.6 server_fd

```
int Server::server_fd = -1  [private]
```

4.7.3.7 should_stop

```
bool Server::should_stop = false  [private]
```

4.7.3.8 storage

```
std::unique_ptr<StorageEngine> Server::storage  [private]
```

The documentation for this class was generated from the following files:

- part-b/src/network_server.h
- part-b/src/network_server.cpp

4.8 StorageEngine Class Reference

```
#include <storage_engine.h>
```

Classes

- struct BatchEntry
- struct DiskEntry

Public Member Functions

- StorageEngine ()
- ~StorageEngine ()
- bool set (const std::string &key, const std::string &value)
- std::string get (const std::string &key)
- bool del (const std::string &key)
- void clear ()
- void force_flush ()
- size_t size () const
- StorageEngine (size_t cache_size=1)
- ~StorageEngine ()
- bool set (const std::string &key, const std::string &value)
- std::string get (const std::string &key)
- bool get (const std::string &key, std::string &value)
- bool del (const std::string &key)
- void clear ()
- void force_flush ()
- size_t size () const
- void sync ()
- size_t pending_write_count () const
- void stop_async_writer ()
- bool put (const std::string &key, const std::string &value)

Private Member Functions

- void load_disk_index ()
- void save_disk_index ()
- void update_disk_index (const std::string &key, size_t offset, size_t size)
- void remove_from_disk_index (const std::string &key)
- void flush_write_buffer ()
- void async_write_worker ()

Private Attributes

- std::unordered_map< std::string, std::string > data_
- std::list< std::string > access_order
- std::mutex mutex_
- size_t pending_writes = 0
- std::map< std::string, DiskEntry > disk_index
- std::vector< BatchEntry > write_buffer
- std::unique_ptr< LRUCache > cache_
- std::unique_ptr< DiskStorage > disk_storage_
- std::queue< std::pair< std::string, std::string > > write_queue_
- std::mutex write_mutex_
- std::condition_variable write_cv_
- std::thread write_thread_
- std::atomic< bool > running_

Static Private Attributes

- static constexpr size_t MAX_KEY_SIZE = 256
- static constexpr size_t MAX_VALUE_SIZE = 1024
- static constexpr size_t MAX_CACHE_SIZE = 10000000
- static constexpr size_t BATCH_SIZE = 1000000
- static constexpr const char * DISK_DIR = "disk_storage"
- static constexpr const char * DATA_FILE = "data.dat"
- static constexpr const char * INDEX_FILE = "index.dat"

4.8.1 Constructor & Destructor Documentation

4.8.1.1 StorageEngine() [1/2]

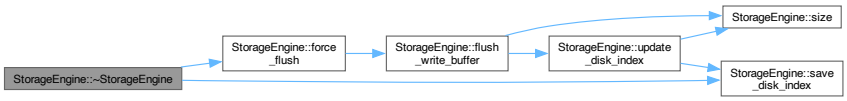
```
StorageEngine::StorageEngine ()
```

Here is the call graph for this function:

4.8.1.2 ~StorageEngine() [1/2]

```
StorageEngine::~StorageEngine ()
```

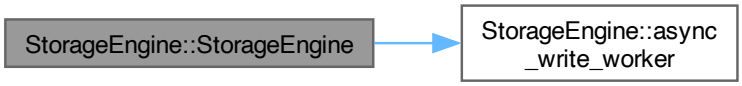
Here is the call graph for this function:



4.8.1.3 StorageEngine() [2/2]

```
StorageEngine::StorageEngine (
            size_t cache_size = 1)  [inline]
```

Here is the call graph for this function:



4.8.1.4 ~StorageEngine() [2/2]

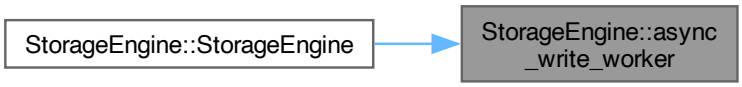
```
StorageEngine::~StorageEngine ()  [inline]
```

4.8.2 Member Function Documentation

4.8.2.1 async_write_worker()

```
void StorageEngine::async_write_worker ()  [inline], [private]
```

Here is the caller graph for this function:



4.8.2.2 clear() [1/2]

```
void StorageEngine::clear ()
```

Here is the caller graph for this function:

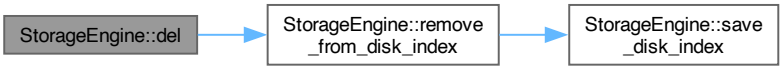
4.8.2.3 clear() [2/2]

```
void StorageEngine::clear ()  [inline]
```

4.8.2.4 del() [1/2]

```
bool StorageEngine::del (
            const std::string & key)
```

Here is the call graph for this function:



Here is the caller graph for this function:

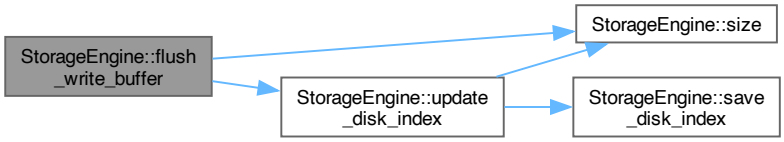
4.8.2.5 del() [2/2]

```
bool StorageEngine::del (
            const std::string & key)  [inline]
```

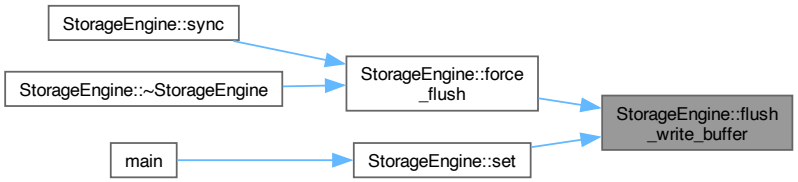
4.8.2.6 flush_write_buffer()

```
void StorageEngine::flush_write_buffer ()  [private]
```

Here is the call graph for this function:



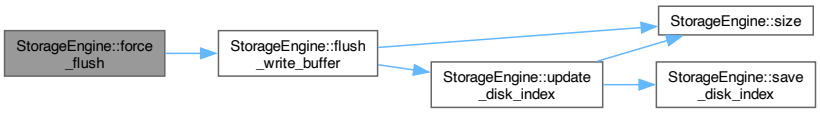
Here is the caller graph for this function:



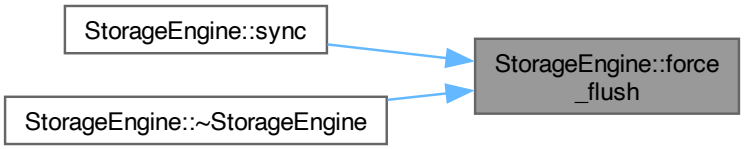
4.8.2.7 force_flush() [1/2]

```
void StorageEngine::force_flush ()
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.8 force_flush() [2/2]

```
void StorageEngine::force_flush ()  [inline]
```

4.8.2.9 get() [1/3]

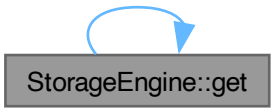
```
std::string StorageEngine::get (
            const std::string & key)
```

Here is the caller graph for this function:

4.8.2.10 get() [2/3]

```
std::string StorageEngine::get (
            const std::string & key)  [inline]
```

Here is the call graph for this function:



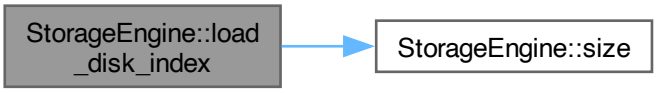
4.8.2.11 get() [3/3]

```
bool StorageEngine::get (
            const std::string & key,
            std::string & value)  [inline]
```

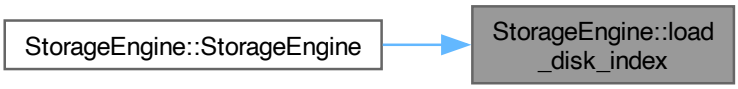
4.8.2.12 load_disk_index()

```
void StorageEngine::load_disk_index ()  [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:



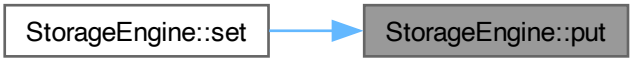
4.8.2.13 pending_write_count()

```
size_t StorageEngine::pending_write_count () const  [inline]
```

4.8.2.14 put()

```
bool StorageEngine::put (
            const std::string & key,
            const std::string & value)  [inline]
```

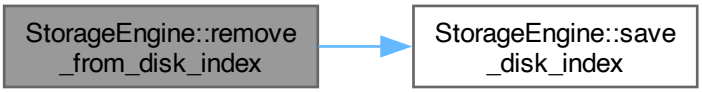
Here is the caller graph for this function:



4.8.2.15 remove_from_disk_index()

```
void StorageEngine::remove_from_disk_index (
            const std::string & key)  [private]
```

Here is the call graph for this function:

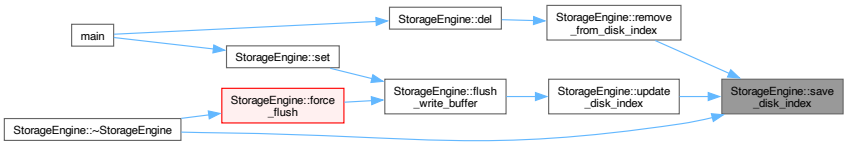


Here is the caller graph for this function:

4.8.2.16 save_disk_index()

```
void StorageEngine::save_disk_index ()  [private]
```

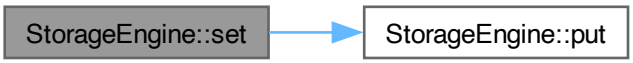
Here is the caller graph for this function:



4.8.2.17 set() [1/2]

```
bool StorageEngine::set (
            const std::string & key,
            const std::string & value)
```

Here is the call graph for this function:

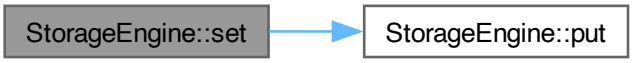


Here is the caller graph for this function:

4.8.2.18 set() [2/2]

```
bool StorageEngine::set (
            const std::string & key,
            const std::string & value)  [inline]
```

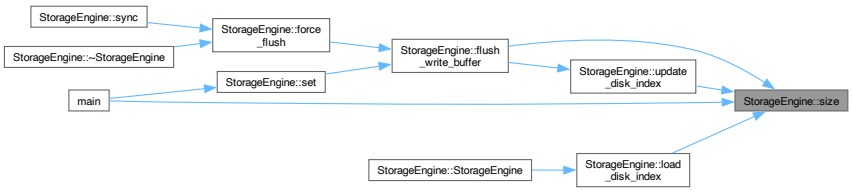
Here is the call graph for this function:



4.8.2.19 size() [1/2]

```
size_t StorageEngine::size () const
```

Here is the caller graph for this function:



4.8.2.20 size() [2/2]

```
size_t StorageEngine::size () const  [inline]
```

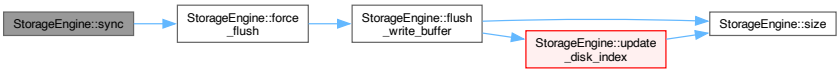
4.8.2.21 stop_async_writer()

```
void StorageEngine::stop_async_writer ()  [inline]
```

4.8.2.22 sync()

```
void StorageEngine::sync ()  [inline]
```

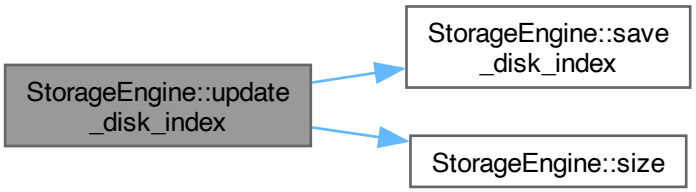
Here is the call graph for this function:



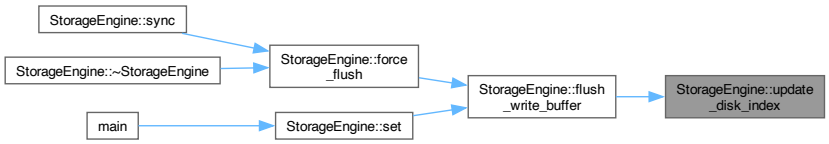
4.8.2.23 update_disk_index()

```
void StorageEngine::update_disk_index (
            const std::string & key,
            size_t offset,
            size_t size)  [private]
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3 Member Data Documentation

4.8.3.1 access_order

```
std::list<std::string> StorageEngine::access_order  [private]
```

4.8.3.2 BATCH_SIZE

```
size_t StorageEngine::BATCH_SIZE = 1000000  [static], [constexpr], [private]
```

4.8.3.3 cache_

```
std::unique_ptr<LRUCache> StorageEngine::cache_  [private]
```

4.8.3.4 data_

```
std::unordered_map<std::string, std::string> StorageEngine::data_  [private]
```

4.8.3.5 DATA_FILE

```
const char* StorageEngine::DATA_FILE = "data.dat"  [static], [constexpr], [private]
```

4.8.3.6 DISK_DIR

```
const char* StorageEngine::DISK_DIR = "disk_storage"  [static], [constexpr], [private]
```

4.8.3.7 disk_index

```
std::map<std::string, DiskEntry> StorageEngine::disk_index  [private]
```

4.8.3.8 disk_storage_

```
std::unique_ptr<DiskStorage> StorageEngine::disk_storage_  [private]
```

4.8.3.9 INDEX_FILE

```
const char* StorageEngine::INDEX_FILE = "index.dat"  [static], [constexpr], [private]
```

4.8.3.10 MAX_CACHE_SIZE

```
size_t StorageEngine::MAX_CACHE_SIZE = 10000000  [static], [constexpr], [private]
```

4.8.3.11 MAX_KEY_SIZE

```
size_t StorageEngine::MAX_KEY_SIZE = 256  [static], [constexpr], [private]
```

4.8.3.12 MAX_VALUE_SIZE

```
size_t StorageEngine::MAX_VALUE_SIZE = 1024  [static], [constexpr], [private]
```

4.8.3.13 mutex_

```
std::mutex StorageEngine::mutex_  [mutable], [private]
```

4.8.3.14 pending_writes

```
size_t StorageEngine::pending_writes = 0  [private]
```

4.8.3.15 running_

```
std::atomic<bool> StorageEngine::running_  [private]
```

4.8.3.16 write_buffer

```
std::vector<BatchEntry> StorageEngine::write_buffer  [private]
```

4.8.3.17 write_cv_

```
std::condition_variable StorageEngine::write_cv_  [private]
```

4.8.3.18 write_mutex_

```
std::mutex StorageEngine::write_mutex_  [private]
```

4.8.3.19 write_queue_

```
std::queue<std::pair<std::string, std::string> > StorageEngine::write_queue_  [private]
```

4.8.3.20 write_thread_

```
std::thread StorageEngine::write_thread_  [private]
```

The documentation for this class was generated from the following files:

- part-a/src/storage_engine.h
- part-b/src/storage_engine.h
- part-a/src/storage_engine.cpp

Chapter 5

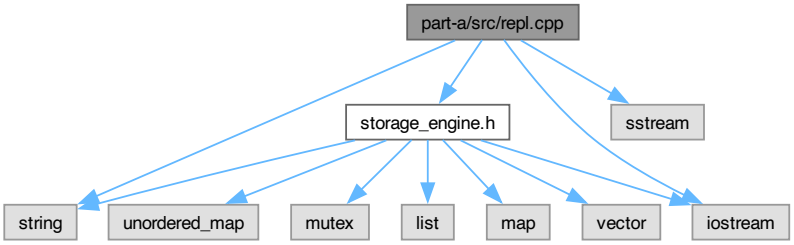
File Documentation

5.1 docs/mainpage.md File Reference

5.2 part-a/src/repl.cpp File Reference

```
#include "storage_engine.h"
#include <iostream>
#include <sstream>
#include <string>
```

Include dependency graph for repl.cpp:



Functions

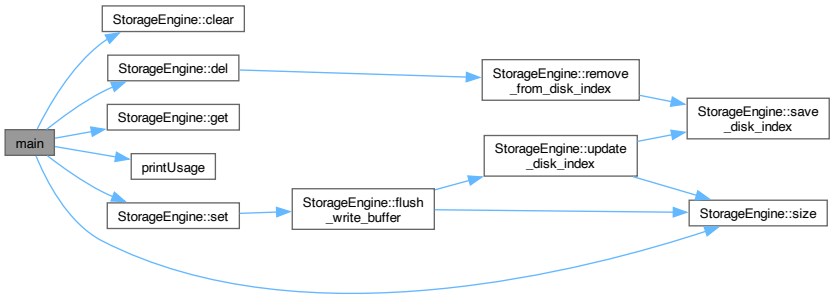
- void printUsage ()
- int main ()

5.2.1 Function Documentation

5.2.1.1 main()

```
int main ()
```

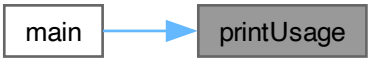
Here is the call graph for this function:



5.2.1.2 printUsage()

```
void printUsage ()
```

Here is the caller graph for this function:

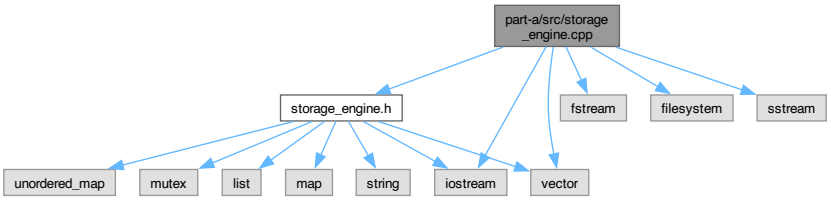


5.3 part-a/src/storage_engine.cpp File Reference

```
#include "storage_engine.h"
#include <fstream>
#include <iostream>
#include <filesystem>
#include <vector>
```

```
#include <sstream>
```

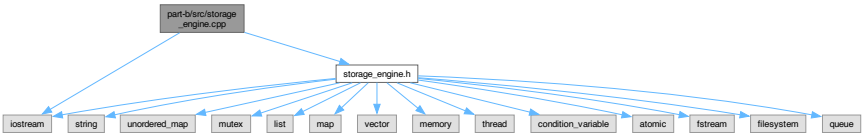
Include dependency graph for storage_engine.cpp:



5.4 part-b/src/storage_engine.cpp File Reference

```
#include "storage_engine.h"
#include <iostream>
```

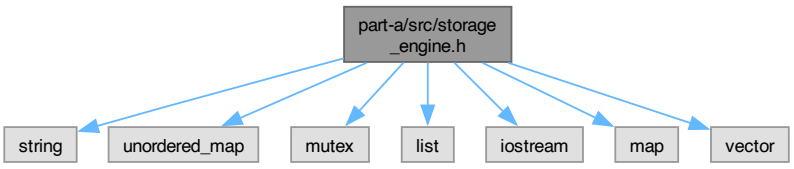
Include dependency graph for storage_engine.cpp:



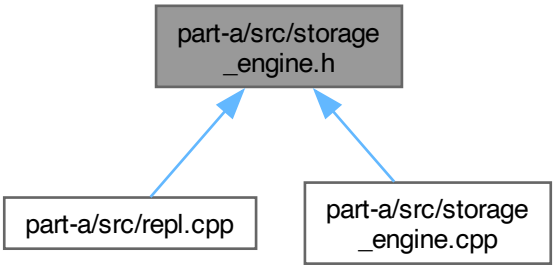
5.5 part-a/src/storage_engine.h File Reference

```
#include <string>
#include <unordered_map>
#include <mutex>
#include <list>
#include <iostream>
#include <map>
#include <vector>
```

Include dependency graph for storage_engine.h:



This graph shows which files directly or indirectly include this file:



Classes

- class StorageEngine
- struct StorageEngine::DiskEntry
- struct StorageEngine::BatchEntry

5.6 storage_engine.h

Go to the documentation of this file.

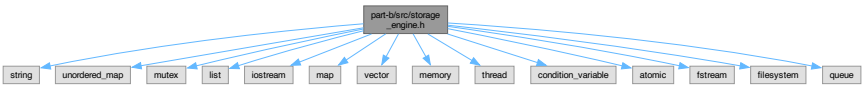
```
00001 #pragma once
00002
00003 #include <string>
00004 #include <unordered_map>
00005 #include <mutex>
00006 #include <list>
00007 #include <iostream>
00008 #include <map>
00009 #include <vector>
00010
00011 class StorageEngine {
00012 public:
00013     StorageEngine();
00014     ~StorageEngine();
00015
00016     bool set(const std::string& key, const std::string& value);
00017     std::string get(const std::string& key);
00018     bool del(const std::string& key);
00019     void clear();  // Clear all data from memory and disk
00020     void force_flush();  // Force flush write buffer
00021     size_t size() const;  // Get total number of entries (in memory + on disk)
00022
00023 private:
00024     static constexpr size_t MAX_KEY_SIZE = 256;
00025     static constexpr size_t MAX_VALUE_SIZE = 1024;
00026     static constexpr size_t MAX_CACHE_SIZE = 10000000;  // 10M entries max in memory
00027     static constexpr size_t BATCH_SIZE = 1000000;  // 1M entries to batch write
00028     static constexpr const char* DISK_DIR = "disk_storage";
00029     static constexpr const char* DATA_FILE = "data.dat";
00030     static constexpr const char* INDEX_FILE = "index.dat";
00031
00032     struct DiskEntry {
00033         size_t offset;
00034         size_t size;
00035     };
00036
00037     struct BatchEntry {
00038         std::string key;
00039         std::string value;
```

```
00040     };
00041
00042     std::unordered_map<std::string, std::string> data_;
00043     std::list<std::string> access_order;  // Track access order for LRU eviction
00044     mutable std::mutex mutex_;  // Make mutex mutable for const methods
00045     size_t pending_writes = 0;  // Track number of pending writes
00046     std::map<std::string, DiskEntry> disk_index;  // Index for disk entries
00047     std::vector<BatchEntry> write_buffer;  // Buffer for batch writes
00048
00049     void load_disk_index();
00050     void save_disk_index();
00051     void update_disk_index(const std::string& key, size_t offset, size_t size);
00052     void remove_from_disk_index(const std::string& key);
00053     void flush_write_buffer();
00054 };
```

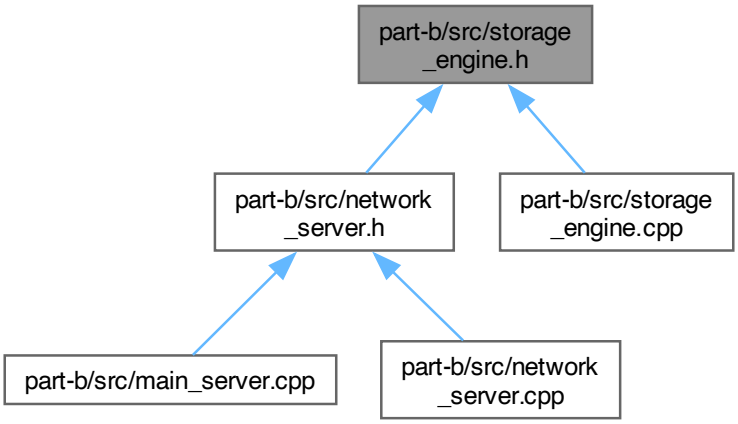
5.7 part-b/src/storage_engine.h File Reference

```
#include <string>
#include <unordered_map>
#include <mutex>
#include <list>
#include <iostream>
#include <map>
#include <vector>
#include <memory>
#include <thread>
#include <condition_variable>
#include <atomic>
#include <fstream>
#include <filesystem>
#include <queue>
```

Include dependency graph for storage_engine.h:



This graph shows which files directly or indirectly include this file:



Classes

- class LRUCache
- struct LRUCache::Node
- class DiskStorage
- class StorageEngine

5.8 storage_engine.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <string>
00004 #include <unordered_map>
00005 #include <mutex>
00006 #include <list>
00007 #include <iostream>
00008 #include <map>
00009 #include <vector>
00010 #include <memory>
00011 #include <thread>
00012 #include <condition_variable>
00013 #include <atomic>
00014 #include <fstream>
00015 #include <filesystem>
00016 #include <queue>
00017 #ifdef __APPLE__
00018 #include <mach-o/dyld.h>
00019 #endif
00020
00021 class LRUCache {
00022 private:
00023     struct Node {
00024         std::string key;
00025         std::string value;
00026         Node* prev;
00027         Node* next;
00028         Node(const std::string& k, const std::string& v)
00029             : key(k), value(v), prev(nullptr), next(nullptr) {}
00030     };
00031
```

```
00032     size_t capacity_;
00033     std::unordered_map<std::string, Node*> cache_;
00034     Node* head_;
00035     Node* tail_;
00036     std::mutex mutex_;
00037 
00038     void move_to_front(Node* node) {
00039         if (node == head_) return;
00040 
00041         if (node == tail_) {
00042             tail_ = node->prev;
00043             tail_->next = nullptr;
00044         } else {
00045             node->prev->next = node->next;
00046             node->next->prev = node->prev;
00047         }
00048 
00049         node->prev = nullptr;
00050         node->next = head_;
00051         head_->prev = node;
00052         head_ = node;
00053     }
00054 
00055     void evict_lru() {
00056         if (tail_) {
00057             cache_.erase(tail_->key);
00058             Node* temp = tail_;
00059             tail_ = tail_->prev;
00060             if (tail_) tail_->next = nullptr;
00061             delete temp;
00062         }
00063     }
00064 
00065 public:
00066     LRUCache(size_t capacity) : capacity_(capacity), head_(nullptr), tail_(nullptr) {}
00067 
00068     ~LRUCache() {
00069         Node* current = head_;
00070         while (current) {
00071             Node* temp = current;
00072             current = current->next;
00073             delete temp;
00074         }
00075     }
00076 
00077     size_t capacity() const { return capacity_; }
00078     size_t size() const { return cache_.size(); }
00079 
00080     bool get(const std::string& key, std::string& value) {
00081         std::lock_guard<std::mutex> lock(mutex_);
00082         auto it = cache_.find(key);
00083         if (it != cache_.end()) {
00084             move_to_front(it->second);
00085             value = it->second->value;
00086             return true;
00087         }
00088         return false;
00089     }
00090 
00091     bool put(const std::string& key, const std::string& value) {
00092         std::lock_guard<std::mutex> lock(mutex_);
00093         auto it = cache_.find(key);
00094         if (it != cache_.end()) {
00095             it->second->value = value;
00096             move_to_front(it->second);
00097             return true;
00098         }
00099 
00100         if (cache_.size() >= capacity_) {
00101             evict_lru();
00102         }
00103 
00104         Node* new_node = new Node(key, value);
00105         cache_[key] = new_node;
00106 
00107         if (!head_) {
00108             head_ = tail_ = new_node;
00109         } else {
00110             new_node->next = head_;
00111             head_->prev = new_node;
00112             head_ = new_node;
00113         }
00114 
00115         return true;
00116     }
00117 
00118     bool remove(const std::string& key) {
```

```
00119         std::lock_guard<std::mutex> lock(mutex_);
00120         auto it = cache_.find(key);
00121         if (it != cache_.end()) {
00122             Node* node = it->second;
00123             if (node == head_) {
00124                 head_ = node->next;
00125                 if (head_) head_->prev = nullptr;
00126                 else tail_ = nullptr;
00127             } else if (node == tail_) {
00128                 tail_ = node->prev;
00129                 if (tail_) tail_->next = nullptr;
00130             } else {
00131                 node->prev->next = node->next;
00132                 node->next->prev = node->prev;
00133             }
00134             cache_.erase(key);
00135             delete node;
00136             return true;
00137         }
00138         return false;
00139     }
00140 };
00141
00142 class DiskStorage {
00143 private:
00144     std::string data_file_;
00145     std::mutex mutex_;
00146     std::unordered_map<std::string, std::string> data_;
00147
00148     std::filesystem::path get_executable_path() {
00149         #ifdef __APPLE__
00150             char path[1024];
00151             uint32_t size = sizeof(path);
00152             if (_NSGetExecutablePath(path, &size) == 0) {
00153                 return std::filesystem::path(path).parent_path();
00154             }
00155         #else
00156             char result[PATH_MAX];
00157             ssize_t count = readlink("/proc/self/exe", result, sizeof(result));
00158             if (count != -1) {
00159                 return std::filesystem::path(result).parent_path();
00160             }
00161         #endif
00162         return std::filesystem::current_path();
00163     }
00164
00165     void ensure_directory_exists() {
00166         std::filesystem::path exe_path = get_executable_path();
00167         std::filesystem::path storage_dir = exe_path / "disk_storage";
00168         std::filesystem::create_directories(storage_dir);
00169     }
00170
00171     void load_data() {
00172         std::ifstream file(data_file_);
00173         if (!file.is_open()) return;
00174
00175         std::string line;
00176         while (std::getline(file, line)) {
00177             size_t pos = line.find('=');
00178             if (pos != std::string::npos) {
00179                 std::string key = line.substr(0, pos);
00180                 std::string value = line.substr(pos + 1);
00181                 data_[key] = value;
00182             }
00183         }
00184         file.close();
00185     }
00186
00187     void save_data() {
00188         std::ofstream file(data_file_);
00189         if (!file.is_open()) return;
00190
00191         for (const auto& [key, value] : data_) {
00192             file « key « "=" « value « "\n";
00193         }
00194         file.close();
00195     }
00196
00197 public:
00198     DiskStorage() {
00199         try {
00200             ensure_directory_exists();
00201             std::filesystem::path exe_path = get_executable_path();
00202             data_file_ = (exe_path / "disk_storage" / "data.txt").string();
00203             load_data();
00204         } catch (const std::exception& e) {
00205             // If loading fails, start with empty data
```

```
00206             data_.clear();
00207         }
00208     }
00209
00210     ~DiskStorage() {
00211         save_data();
00212     }
00213
00214     bool get(const std::string& key, std::string& value) {
00215         std::lock_guard<std::mutex> lock(mutex_);
00216         auto it = data_.find(key);
00217         if (it != data_.end()) {
00218             value = it->second;
00219             return true;
00220         }
00221         return false;
00222     }
00223
00224     bool put(const std::string& key, const std::string& value) {
00225         std::lock_guard<std::mutex> lock(mutex_);
00226         data_[key] = value;
00227         save_data();
00228         return true;
00229     }
00230
00231     void remove(const std::string& key) {
00232         std::lock_guard<std::mutex> lock(mutex_);
00233         data_.erase(key);
00234         save_data();
00235     }
00236 };
00237
00238 class StorageEngine {
00239 public:
00240     StorageEngine(size_t cache_size = 1)
00241         : cache_(std::make_unique<LRUCache>(cache_size))
00242         , running_(false) {
00243         try {
00244             disk_storage_ = std::make_unique<DiskStorage>();
00245             running_ = true;
00246             write_thread_ = std::thread(&StorageEngine::async_write_worker, this);
00247         } catch (const std::exception& e) {
00248             running_ = false;
00249             throw;
00250         }
00251     }
00252
00253     ~StorageEngine() {
00254         if (running_) {
00255             running_ = false;
00256             write_cv_.notify_one();
00257             if (write_thread_.joinable()) {
00258                 write_thread_.join();
00259             }
00260         }
00261     }
00262
00263     bool set(const std::string& key, const std::string& value) {
00264         return put(key, value);
00265     }
00266
00267     std::string get(const std::string& key) {
00268         std::string value;
00269         if (get(key, value)) {
00270             return value;
00271         }
00272         return "";
00273     }
00274
00275     bool get(const std::string& key, std::string& value) {
00276         // First check cache
00277         if (cache_->get(key, value)) {
00278             return true;
00279         }
00280
00281         // If not in cache, check disk storage
00282         if (disk_storage_->get(key, value)) {
00283             // Add to cache
00284             cache_->put(key, value);
00285             return true;
00286         }
00287         return false;
00288     }
00289
00290     bool del(const std::string& key) {
00291         try {
00292             // First check if key exists in either cache or disk
```

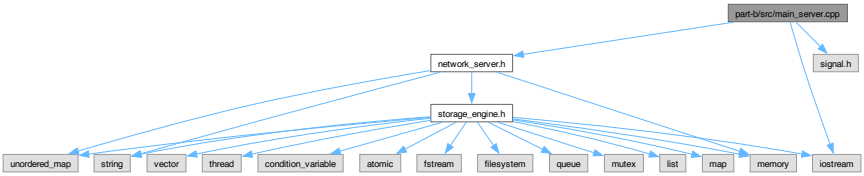
```
00293             std::string value;
00294             bool exists = false;
00295 
00296             // Check cache first
00297             if (cache_->get(key, value)) {
00298                 exists = true;
00299                 cache_->remove(key);
00300             }
00301 
00302             // Then check disk storage
00303             if (disk_storage_->get(key, value)) {
00304                 exists = true;
00305                 disk_storage_->remove(key);
00306             }
00307 
00308             return exists;
00309         } catch (const std::exception& e) {
00310             return false;
00311         }
00312     }
00313 
00314     void clear() {
00315         cache_ = std::make_unique<LRUCache>(cache_->capacity());
00316         disk_storage_ = std::make_unique<DiskStorage>();
00317     }
00318 
00319     void force_flush() {
00320         std::lock_guard<std::mutex> lock(write_mutex_);
00321         while (!write_queue_.empty()) {
00322             auto [key, value] = write_queue_.front();
00323             write_queue_.pop();
00324             disk_storage_->put(key, value);
00325         }
00326     }
00327 
00328     size_t size() const {
00329         return cache_->size();
00330     }
00331 
00332     void sync() {
00333         force_flush();
00334     }
00335 
00336     size_t pending_write_count() const {
00337         std::lock_guard<std::mutex> lock(const_cast<std::mutex&>(write_mutex_));
00338         return write_queue_.size();
00339     }
00340 
00341     void stop_async_writer() {
00342         running_ = false;
00343         write_cv_.notify_one();
00344         if (write_thread_.joinable()) {
00345             write_thread_.join();
00346         }
00347     }
00348 
00349     bool put(const std::string& key, const std::string& value) {
00350         // First try to put in cache
00351         if (cache_->put(key, value)) {
00352             // If successful, queue async write to disk
00353             {
00354                 std::lock_guard<std::mutex> lock(write_mutex_);
00355                 write_queue_.push({key, value});
00356             }
00357             write_cv_.notify_one();
00358             return true;
00359         }
00360 
00361         // If cache is full, write directly to disk
00362         return disk_storage_->put(key, value);
00363     }
00364 
00365 private:
00366     void async_write_worker() {
00367         while (running_) {
00368             std::unique_lock<std::mutex> lock(write_mutex_);
00369             write_cv_.wait(lock, [this] {
00370                 return !running_ || !write_queue_.empty();
00371             });
00372 
00373             if (!running_) break;
00374 
00375             auto [key, value] = write_queue_.front();
00376             write_queue_.pop();
00377             lock.unlock();
00378 
00379             disk_storage_->put(key, value);
```

```
00380         }
00381     }
00382
00383     std::unique_ptr<LRUCache> cache_;
00384     std::unique_ptr<DiskStorage> disk_storage_;
00385     std::queue<std::pair<std::string, std::string» write_queue_;
00386     std::mutex write_mutex_;
00387     std::condition_variable write_cv_;
00388     std::thread write_thread_;
00389     std::atomic<bool> running_;
00390 };
```

5.9 part-b/src/main_server.cpp File Reference

```
#include "network_server.h"
#include <iostream>
#include <signal.h>
```

Include dependency graph for main_server.cpp:



Functions

- void signal_handler (int signum)
- int main ()

Variables

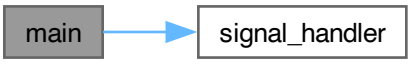
- Server * g_server = nullptr

5.9.1 Function Documentation

5.9.1.1 main()

```
int main ()
```

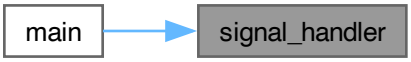
Here is the call graph for this function:



5.9.1.2 signal_handler()

```
void signal_handler (
            int signum)
```

Here is the caller graph for this function:



5.9.2 Variable Documentation

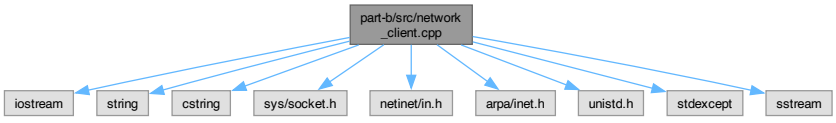
5.9.2.1 g_server

```
Server* g_server = nullptr
```

5.10 part-b/src/network_client.cpp File Reference

```
#include <iostream>
#include <string>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdexcept>
#include <sstream>
```

Include dependency graph for network_client.cpp:



Classes

- class NetworkClient

Functions

- int main (int argc, char *argv[ ])

5.10.1 Function Documentation

5.10.1.1 main()

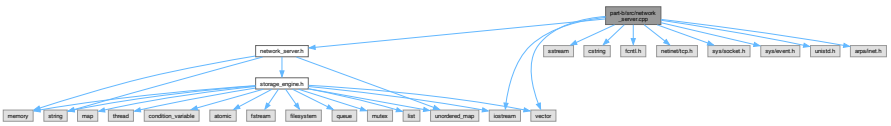
```
int main (
            int argc,
            char * argv[])
```

Here is the call graph for this function:

5.11 part-b/src/network_server.cpp File Reference

```
#include "network_server.h"
#include <iostream>
#include <sstream>
#include <vector>
#include <cstring>
#include <fcntl.h>
#include <netinet/tcp.h>
#include <sys/socket.h>
#include <sys/event.h>
#include <unistd.h>
#include <arpa/inet.h>
```

Include dependency graph for network_server.cpp:



Macros

- #define TCP_NODELAY 1

5.11.1 Macro Definition Documentation

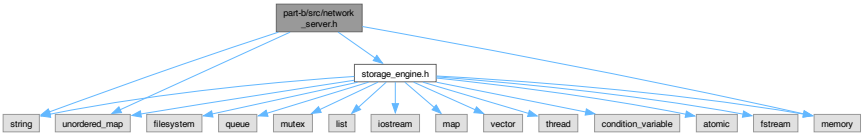
5.11.1.1 TCP_NODELAY

```
#define TCP_NODELAY 1
```

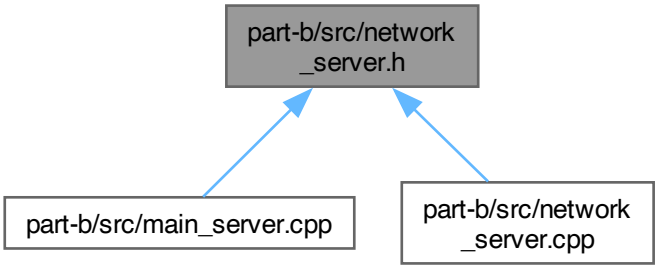
5.12 part-b/src/network_server.h File Reference

```
#include "storage_engine.h"
#include <memory>
#include <string>
#include <unordered_map>
```

Include dependency graph for network_server.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Server

5.13 network_server.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "storage_engine.h"
00004 #include <memory>
00005 #include <string>
00006 #include <unordered_map>
00007
00008 class Server {
00009 public:
00010     Server();
00011     ~Server();
00012
00013     void run();
00014     void stop();
00015
```

```
00016 private:
00017     static constexpr int PORT = 9001;
00018     static constexpr int LISTEN_BACKLOG = 128;
00019     static constexpr int MAX_EVENTS = 1024;
00020
00021     int server_fd = -1;
00022     int kq = -1;
00023     bool should_stop = false;
00024     std::unique_ptr<StorageEngine> storage;
00025     std::unordered_map<int, std::string> client_buffers;
00026
00027     void setup_server();
00028     void setup_kqueue();
00029     void set_nonblocking(int fd);
00030     void handle_new_connection();
00031     void handle_client_data(int client_fd);
00032     std::string process_command(const std::string& command);
00033     std::string encode_resp(const std::string& response);
00034 };
```

Index

~DiskStorage
  DiskStorage, 15
~LRUCache
  LRUCache, 19
~NetworkClient
  NetworkClient, 23
~Server
  Server, 26
~StorageEngine
  StorageEngine, 33

access_order
  StorageEngine, 42
async_write_worker
  StorageEngine, 34

BATCH_SIZE
  StorageEngine, 42
BLINK DB Documentation, 1

cache_
  LRUCache, 22
  StorageEngine, 42
capacity
  LRUCache, 20
capacity_
  LRUCache, 22
clear
  StorageEngine, 34
client_buffers
  Server, 30

data_
  DiskStorage, 18
  StorageEngine, 42
DATA_FILE
  StorageEngine, 42
data_file_
  DiskStorage, 18
del
  StorageEngine, 34, 35
DISK_DIR
  StorageEngine, 42
disk_index
  StorageEngine, 43
disk_storage_
  StorageEngine, 43
DiskStorage, 14
  ~DiskStorage, 15
  data_, 18
  data_file_, 18
  DiskStorage, 15
  ensure_directory_exists, 15
  get, 16
  get_executable_path, 16
  load_data, 16
  mutex_, 18
  put, 16
  remove, 17
  save_data, 17
docs/mainpage.md, 45

encode_resp
  Server, 27
ensure_directory_exists
  DiskStorage, 15
evict_lru
  LRUCache, 20

flush_write_buffer
  StorageEngine, 35
force_flush
  StorageEngine, 36

g_server
  main_server.cpp, 56
get
  DiskStorage, 16
  LRUCache, 20
  StorageEngine, 36, 37
get_executable_path
  DiskStorage, 16

handle_client_data
  Server, 27
handle_new_connection
  Server, 27
head_
  LRUCache, 22

INDEX_FILE
  StorageEngine, 43

key
  LRUCache::Node, 25
  StorageEngine::BatchEntry, 13
kq
  Server, 30

LISTEN_BACKLOG
  Server, 31

load_data
DiskStorage, 16
load_disk_index
StorageEngine, 37
LRUCache, 18
~LRUCache, 19
cache_, 22
capacity, 20
capacity_, 22
evict_lru, 20
get, 20
head_, 22
LRUCache, 19
move_to_front, 20
mutex_, 22
put, 21
remove, 21
size, 21
tail_, 22
LRUCache::Node, 24
key, 25
next, 25
Node, 25
prev, 25
value, 25

main
main_server.cpp, 55
network_client.cpp, 57
repl.cpp, 46
main_server.cpp
g_server, 56
main, 55
signal_handler, 55
MAX_CACHE_SIZE
StorageEngine, 43
MAX_EVENTS
Server, 31
MAX_KEY_SIZE
StorageEngine, 43
MAX_VALUE_SIZE
StorageEngine, 43
move_to_front
LRUCache, 20
mutex_
DiskStorage, 18
LRUCache, 22
StorageEngine, 43

network_client.cpp
main, 57
network_server.cpp
TCP_NODELAY, 57
NetworkClient, 22
~NetworkClient, 23
NetworkClient, 23
parse_response, 23
send_command, 23
server_addr, 24
sock, 24
next
LRUCache::Node, 25
Node
LRUCache::Node, 25

offset
StorageEngine::DiskEntry, 14

parse_response
NetworkClient, 23
part-a/src/repl.cpp, 45
part-a/src/storage_engine.cpp, 46
part-a/src/storage_engine.h, 47, 48
part-b/src/main_server.cpp, 55
part-b/src/network_client.cpp, 56
part-b/src/network_server.cpp, 57
part-b/src/network_server.h, 58
part-b/src/storage_engine.cpp, 47
part-b/src/storage_engine.h, 49, 50
pending_write_count
StorageEngine, 38
pending_writes
StorageEngine, 43
PORT
Server, 31
prev
LRUCache::Node, 25
printUsage
repl.cpp, 46
process_command
Server, 28
put
DiskStorage, 16
LRUCache, 21
StorageEngine, 38

remove
DiskStorage, 17
LRUCache, 21
remove_from_disk_index
StorageEngine, 38
repl.cpp
main, 46
printUsage, 46
run
Server, 28
running_
StorageEngine, 43

save_data
DiskStorage, 17
save_disk_index
StorageEngine, 39
send_command
NetworkClient, 23
Server, 25
~Server, 26
client_buffers, 30

encode_resp, 27
handle_client_data, 27
handle_new_connection, 27
kq, 30
LISTEN_BACKLOG, 31
MAX_EVENTS, 31
PORT, 31
process_command, 28
run, 28
Server, 26
server_fd, 31
set_nonblocking, 29
setup_kqueue, 29
setup_server, 29
should_stop, 31
stop, 30
storage, 31
server_addr
NetworkClient, 24
server_fd
Server, 31
set
StorageEngine, 39, 40
set_nonblocking
Server, 29
setup_kqueue
Server, 29
setup_server
Server, 29
should_stop
Server, 31
signal_handler
main_server.cpp, 55
size
LRUCache, 21
StorageEngine, 40, 41
StorageEngine::DiskEntry, 14
sock
NetworkClient, 24
stop
Server, 30
stop_async_writer
StorageEngine, 41
storage
Server, 31
StorageEngine, 31
~StorageEngine, 33
access_order, 42
async_write_worker, 34
BATCH_SIZE, 42
cache_, 42
clear, 34
data_, 42
DATA_FILE, 42
del, 34, 35
DISK_DIR, 42
disk_index, 43
disk_storage_, 43
flush_write_buffer, 35
force_flush, 36
get, 36, 37
INDEX_FILE, 43
load_disk_index, 37
MAX_CACHE_SIZE, 43
MAX_KEY_SIZE, 43
MAX_VALUE_SIZE, 43
mutex_, 43
pending_write_count, 38
pending_writes, 43
put, 38
remove_from_disk_index, 38
running_, 43
save_disk_index, 39
set, 39, 40
size, 40, 41
stop_async_writer, 41
StorageEngine, 33
sync, 41
update_disk_index, 41
write_buffer, 43
write_cv_, 44
write_mutex_, 44
write_queue_, 44
write_thread_, 44
StorageEngine::BatchEntry, 13
key, 13
value, 13
StorageEngine::DiskEntry, 13
offset, 14
size, 14
sync
StorageEngine, 41

tail_
LRUCache, 22
TCP_NODELAY
network_server.cpp, 57

update_disk_index
StorageEngine, 41

value
LRUCache::Node, 25
StorageEngine::BatchEntry, 13

write_buffer
StorageEngine, 43
write_cv_
StorageEngine, 44
write_mutex_
StorageEngine, 44
write_queue_
StorageEngine, 44
write_thread_
StorageEngine, 44