

BlinkDB - Part B

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 BlinkDB Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 BlinkDB()	6
3.1.2.2 ~BlinkDB()	6
3.1.3 Member Function Documentation	7
3.1.3.1 clearPersistenceFile()	7
3.1.3.2 del()	7
3.1.3.3 flushToDiskAsync()	7
3.1.3.4 flushToDiskPeriodically()	7
3.1.3.5 get()	7
3.1.3.6 loadFromFile()	8
3.1.3.7 persistToFile()	8
3.1.3.8 restoreFromDisk()	8
3.1.3.9 set()	8
3.1.3.10 updateLRU()	9
3.1.4 Member Data Documentation	9
3.1.4.1 db_mutex	9
3.1.4.2 dirty	9
3.1.4.3 evicted_keys	9
3.1.4.4 lru_keys	10
3.1.4.5 lru_map	10
3.1.4.6 max_cache_size	10
3.1.4.7 persistence_file	10
3.1.4.8 store	10
3.2 BlinkServer Class Reference	10
3.2.1 Detailed Description	12
3.2.2 Constructor & Destructor Documentation	12
3.2.2.1 BlinkServer()	12
3.2.2.2 ~BlinkServer()	12
3.2.3 Member Function Documentation	12
3.2.3.1 decodeCommand()	12
3.2.3.2 encodeBulkString()	13
3.2.3.3 encodeError()	13
3.2.3.4 encodeInteger()	14
3.2.3.5 encodeSimpleString()	14

3.2.3.6 handleClientConnections()	14
3.2.3.7 handleClientRead()	14
3.2.3.8 handleCommand()	15
3.2.3.9 processDel()	15
3.2.3.10 processGet()	16
3.2.3.11 processSet()	16
3.2.3.12 setupServer()	17
3.2.3.13 start()	17
3.2.4 Member Data Documentation	17
3.2.4.1 address	17
3.2.4.2 database	17
3.2.4.3 MAX_CLIENTS	18
3.2.4.4 PORT	18
3.2.4.5 server_fd	18
3.3 LoadBalancer Class Reference	18
3.3.1 Detailed Description	19
3.3.2 Constructor & Destructor Documentation	19
3.3.2.1 LoadBalancer()	19
3.3.2.2 ~LoadBalancer()	20
3.3.3 Member Function Documentation	20
3.3.3.1 connectToBackend()	20
3.3.3.2 handleClient()	20
3.3.3.3 setupServer()	21
3.3.3.4 start()	21
3.3.4 Member Data Documentation	21
3.3.4.1 address	21
3.3.4.2 current_server	21
3.3.4.3 MAX_CLIENTS	21
3.3.4.4 PORT	22
3.3.4.5 server1_ip	22
3.3.4.6 server1_port	22
3.3.4.7 server2_ip	22
3.3.4.8 server2_port	22
3.3.4.9 server_fd	22
4 File Documentation	23
4.1 src/blink_server.cpp File Reference	23
4.2 src/blink_server.h File Reference	23
4.2.1 Detailed Description	24
4.2.2 Macro Definition Documentation	24
4.2.2.1 __BLINK_SERVER_H	24
4.3 src/blinkdb.cpp File Reference	24

4.3.1 Detailed Description	24
4.4 src/blinkdb.h File Reference	25
4.4.1 Detailed Description	25
4.4.2 Macro Definition Documentation	25
4.4.2.1 COMPACTION_THRESHOLD	26
4.4.2.2 FLUSH_FILE	26
4.4.2.3 MAX_CAPACITY	26
4.4.2.4 VALUE_SIZE	26
4.5 src/load_balancer.cpp File Reference	26
4.5.1 Detailed Description	27
4.5.2 Function Documentation	27
4.5.2.1 main()	27
4.6 src/main.cpp File Reference	27
4.6.1 Detailed Description	28
4.6.2 Function Documentation	28
4.6.2.1 main()	28
Index	29

Chapter 1

Class Index

1.1 Class List

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

BlinkDB	An in-memory key-value database with LRU caching and disk persistence	5
BlinkServer	Implements a Redis-compatible server using the RESP-2 protocol	10
LoadBalancer	Implements a round-robin load balancer for multiple backend servers	18

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

src/ blink_server.cpp	
Implementation of the BlinkServer class	23
src/ blink_server.h	
Header file for the BlinkServer class implementing a Redis-like server	23
src/ blinkdb.cpp	
Implementation of the BlinkDB class	24
src/ blinkdb.h	
Header file for the BlinkDB in-memory database with LRU caching	25
src/ load_balancer.cpp	
Implementation of a load balancer for BlinkDB servers	26
src/ main.cpp	
Main entry point for the BlinkServer application	27

Chapter 3

Class Documentation

3.1 BlinkDB Class Reference

An in-memory key-value database with LRU caching and disk persistence.

```
#include <blinkdb.h>
```

Public Member Functions

- [BlinkDB](#) ()
Constructor.
- [~BlinkDB](#) ()
Destructor.
- void [set](#) (const std::string &key, const std::string &value)
Sets a key-value pair in the database.
- std::string [get](#) (const std::string &key)
Retrieves a value by key.
- bool [del](#) (const std::string &key)
Deletes a key-value pair from the database.
- void [persistToFile](#) ()
Writes all in-memory data to disk.
- void [clearPersistenceFile](#) ()
Deletes the persistence file.
- void [flushToDiskPeriodically](#) ()
Background thread function that periodically flushes data to disk.
- void [flushToDiskAsync](#) ()
Asynchronously flushes data to disk.

Private Member Functions

- void [loadFromFile](#) ()
Loads data from persistence file into memory.
- void [updateLRU](#) (const std::string &key)
Updates the LRU status of a key.
- void [restoreFromDisk](#) (const std::string &key)
Restores an evicted key from disk.

Private Attributes

- `std::unordered_map< std::string, std::string >` [store](#)
Main storage for key-value pairs.
- `std::list< std::string >` [lru_keys](#)
List maintaining LRU order of keys.
- `std::unordered_map< std::string, std::list< std::string >::iterator >` [lru_map](#)
Map for quick access to keys' positions in the LRU list.
- `std::unordered_map< std::string, bool >` [evicted_keys](#)
Set of keys that have been evicted from memory but exist on disk.
- `std::shared_mutex` [db_mutex](#)
Mutex for thread-safe access to the database.
- `const size_t` [max_cache_size](#) = [MAX_CAPACITY](#)
Maximum number of items to keep in memory.
- `const std::string` [persistence_file](#) = [FLUSH_FILE](#)
Path to the persistence file.
- `bool` [dirty](#) = false
Flag indicating whether data has been modified since last flush.

3.1.1 Detailed Description

An in-memory key-value database with LRU caching and disk persistence.

[BlinkDB](#) implements a simple key-value store with an LRU (Least Recently Used) eviction policy. It provides persistence by periodically flushing data to disk and can restore evicted keys from disk when requested.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BlinkDB()

```
BlinkDB::BlinkDB ( )
```

Constructor.

Constructor implementation.

Initializes the database and starts the background flush thread

Loads existing data from disk and starts the background flush thread

3.1.2.2 ~BlinkDB()

```
BlinkDB::~BlinkDB ( )
```

Destructor.

Destructor implementation.

Ensures any pending changes are persisted to disk

Ensures any unsaved changes are written to disk

3.1.3 Member Function Documentation

3.1.3.1 clearPersistenceFile()

```
void BlinkDB::clearPersistenceFile ( )
```

Deletes the persistence file.

Removes the persistence file from disk.

3.1.3.2 del()

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

Deletes a key-value pair from the database.

Parameters

<i>key</i>	The key to delete
------------	-------------------

Returns

true if the key was found and deleted, false otherwise

3.1.3.3 flushToDiskAsync()

```
void BlinkDB::flushToDiskAsync ( )
```

Asynchronously flushes data to disk.

3.1.3.4 flushToDiskPeriodically()

```
void BlinkDB::flushToDiskPeriodically ( )
```

Background thread function that periodically flushes data to disk.

3.1.3.5 get()

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

Retrieves a value by key.

Parameters

<i>key</i>	The key to look up
------------	--------------------

Returns

The value associated with the key, or "NULL" if not found

3.1.3.6 loadFromFile()

```
void BlinkDB::loadFromFile ( ) [private]
```

Loads data from persistence file into memory.

3.1.3.7 persistToFile()

```
void BlinkDB::persistToFile ( )
```

Writes all in-memory data to disk.

3.1.3.8 restoreFromDisk()

```
void BlinkDB::restoreFromDisk (
    const std::string & key ) [private]
```

Restores an evicted key from disk.

Parameters

<i>key</i>	The key to restore
------------	--------------------

3.1.3.9 set()

```
void BlinkDB::set (
    const std::string & key,
    const std::string & value )
```

Sets a key-value pair in the database.

Parameters

<i>key</i>	The key to set
<i>value</i>	The value to associate with the key

3.1.3.10 updateLRU()

```
void BlinkDB::updateLRU (
    const std::string & key ) [private]
```

Updates the LRU status of a key.

Parameters

<i>key</i>	The key to update in the LRU cache
<i>key</i>	The key to update in the LRU cache

Moves the key to the front of the LRU list and handles eviction if needed

3.1.4 Member Data Documentation**3.1.4.1 db_mutex**

```
std::shared_mutex BlinkDB::db_mutex [mutable], [private]
```

Mutex for thread-safe access to the database.

3.1.4.2 dirty

```
bool BlinkDB::dirty = false [private]
```

Flag indicating whether data has been modified since last flush.

3.1.4.3 evicted_keys

```
std::unordered_map<std::string, bool> BlinkDB::evicted_keys [private]
```

Set of keys that have been evicted from memory but exist on disk.

3.1.4.4 lru_keys

```
std::list<std::string> BlinkDB::lru_keys [private]
```

List maintaining LRU order of keys.

3.1.4.5 lru_map

```
std::unordered_map<std::string, std::list<std::string>::iterator> BlinkDB::lru_map [private]
```

Map for quick access to keys' positions in the LRU list.

3.1.4.6 max_cache_size

```
const size_t BlinkDB::max_cache_size = MAX_CAPACITY [private]
```

Maximum number of items to keep in memory.

3.1.4.7 persistence_file

```
const std::string BlinkDB::persistence_file = FLUSH_FILE [private]
```

Path to the persistence file.

3.1.4.8 store

```
std::unordered_map<std::string, std::string> BlinkDB::store [private]
```

Main storage for key-value pairs.

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

- [src/blinkdb.h](#)
- [src/blinkdb.cpp](#)

3.2 BlinkServer Class Reference

Implements a Redis-compatible server using the RESP-2 protocol.

```
#include <blink_server.h>
```


Public Member Functions

- [BlinkServer](#) ()
Constructor.
- [~BlinkServer](#) ()
Destructor.
- void [start](#) ()
Starts the server.

Private Member Functions

- std::string [encodeSimpleString](#) (const std::string &msg)
Encodes a simple string in RESP-2 format.
- std::string [encodeBulkString](#) (const std::string &msg)
Encodes a bulk string in RESP-2 format.
- std::string [encodeInteger](#) (int value)
Encodes an integer in RESP-2 format.
- std::string [encodeError](#) (const std::string &msg)
Encodes an error message in RESP-2 format.
- std::vector< std::string > [decodeCommand](#) (const std::string &raw_input)
Decodes a RESP-2 command from raw input.
- std::string [handleCommand](#) (const std::vector< std::string > &command)
Handles a decoded command.
- std::string [processSet](#) (const std::vector< std::string > &args)
Processes a SET command.
- std::string [processGet](#) (const std::vector< std::string > &args)
Processes a GET command.
- std::string [processDel](#) (const std::vector< std::string > &args)
Processes a DEL command.
- void [setupServer](#) ()
Sets up the server socket.
- void [handleClientConnections](#) ()
Handles client connections using epoll.
- void [handleClientRead](#) (int client_socket)
Handles a read event from a client.

Private Attributes

- int [server_fd](#)
Socket file descriptor for the server.
- struct sockaddr_in [address](#)
Server address structure.
- std::unique_ptr< [BlinkDB](#) > [database](#)
Database instance for storing key-value pairs.

Static Private Attributes

- static const int [PORT](#) = 9001
Port number the server listens on.
- static const int [MAX_CLIENTS](#) = 1500
Maximum number of simultaneous client connections.

3.2.1 Detailed Description

Implements a Redis-compatible server using the RESP-2 protocol.

This class provides a server that listens for client connections and processes Redis-compatible commands using the RESP-2 protocol. It uses epoll for efficient handling of multiple client connections.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 BlinkServer()

```
BlinkServer::BlinkServer ( )
```

Constructor.

Constructor implementation.

Initializes the database and sets up the server socket.

3.2.2.2 ~BlinkServer()

```
BlinkServer::~~BlinkServer ( )
```

Destructor.

Destructor implementation.

Closes the server socket.

3.2.3 Member Function Documentation

3.2.3.1 decodeCommand()

```
std::vector< std::string > BlinkServer::decodeCommand (
    const std::string & raw_input ) [private]
```

Decodes a RESP-2 command from raw input.

Parameters

<i>raw_input</i>	The raw input string to decode
------------------	--------------------------------

Returns

Vector of command arguments

Parameters

<i>raw_input</i>	The raw input string to decode
------------------	--------------------------------

Returns

Vector of command arguments

Parses RESP-2 protocol formatted commands into a vector of strings.

3.2.3.2 encodeBulkString()

```
std::string BlinkServer::encodeBulkString (
    const std::string & msg ) [private]
```

Encodes a bulk string in RESP-2 format.

Parameters

<i>msg</i>	The string to encode
------------	----------------------

Returns

The RESP-2 encoded bulk string

3.2.3.3 encodeError()

```
std::string BlinkServer::encodeError (
    const std::string & msg ) [private]
```

Encodes an error message in RESP-2 format.

Parameters

<i>msg</i>	The error message to encode
------------	-----------------------------

Returns

The RESP-2 encoded error message

3.2.3.4 encodeInteger()

```
std::string BlinkServer::encodeInteger (
    int value ) [private]
```

Encodes an integer in RESP-2 format.

Parameters

<i>value</i>	The integer to encode
--------------	-----------------------

Returns

The RESP-2 encoded integer

3.2.3.5 encodeSimpleString()

```
std::string BlinkServer::encodeSimpleString (
    const std::string & msg ) [private]
```

Encodes a simple string in RESP-2 format.

Parameters

<i>msg</i>	The string to encode
------------	----------------------

Returns

The RESP-2 encoded simple string

3.2.3.6 handleClientConnections()

```
void BlinkServer::handleClientConnections ( ) [private]
```

Handles client connections using epoll.

Main event loop that accepts new connections and processes client requests using epoll for efficient I/O multiplexing.

3.2.3.7 handleClientRead()

```
void BlinkServer::handleClientRead (
    int client_socket ) [private]
```

Handles a read event from a client.

Parameters

<i>client_socket</i>	The client socket file descriptor
----------------------	-----------------------------------

Reads data from the client, processes the command, and sends the response.

3.2.3.8 handleCommand()

```
std::string BlinkServer::handleCommand (  
    const std::vector< std::string > & command ) [private]
```

Handles a decoded command.

Parameters

<i>command</i>	Vector of command arguments
----------------	-----------------------------

Returns

RESP-2 encoded response

Parameters

<i>command</i>	Vector of command arguments
----------------	-----------------------------

Returns

RESP-2 encoded response

Routes the command to the appropriate handler based on the command type.

3.2.3.9 processDel()

```
std::string BlinkServer::processDel (  
    const std::vector< std::string > & args ) [private]
```

Processes a DEL command.

Parameters

<i>args</i>	Command arguments
-------------	-------------------

Returns

RESP-2 encoded response

Parameters

<i>args</i>	Command arguments
-------------	-------------------

Returns

RESP-2 encoded response

Deletes a key-value pair from the database.

3.2.3.10 processGet()

```
std::string BlinkServer::processGet (
    const std::vector< std::string > & args ) [private]
```

Processes a GET command.

Parameters

<i>args</i>	Command arguments
-------------	-------------------

Returns

RESP-2 encoded response

Parameters

<i>args</i>	Command arguments
-------------	-------------------

Returns

RESP-2 encoded response

Retrieves a value by key from the database.

3.2.3.11 processSet()

```
std::string BlinkServer::processSet (
    const std::vector< std::string > & args ) [private]
```

Processes a SET command.

Parameters

<i>args</i>	Command arguments
-------------	-------------------

Returns

RESP-2 encoded response

Parameters

<i>args</i>	Command arguments
-------------	-------------------

Returns

RESP-2 encoded response

Sets a key-value pair in the database.

3.2.3.12 setupServer()

```
void BlinkServer::setupServer ( ) [private]
```

Sets up the server socket.

Creates and configures the server socket, binds it to the port, and starts listening for connections.

3.2.3.13 start()

```
void BlinkServer::start ( )
```

Starts the server.

Begins listening for and handling client connections.

3.2.4 Member Data Documentation**3.2.4.1 address**

```
struct sockaddr_in BlinkServer::address [private]
```

Server address structure.

3.2.4.2 database

```
std::unique_ptr<BlinkDB> BlinkServer::database [private]
```

Database instance for storing key-value pairs.

3.2.4.3 MAX_CLIENTS

```
const int BlinkServer::MAX_CLIENTS = 1500 [static], [private]
```

Maximum number of simultaneous client connections.

3.2.4.4 PORT

```
const int BlinkServer::PORT = 9001 [static], [private]
```

Port number the server listens on.

3.2.4.5 server_fd

```
int BlinkServer::server_fd [private]
```

Socket file descriptor for the server.

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

- [src/blink_server.h](#)
- [src/blink_server.cpp](#)

3.3 LoadBalancer Class Reference

Implements a round-robin load balancer for multiple backend servers.

Public Member Functions

- [LoadBalancer](#) (int port, const std::string &s1_ip, int s1_port, const std::string &s2_ip, int s2_port)
Constructor.
- [~LoadBalancer](#) ()
Destructor.
- void [setupServer](#) ()
Sets up the server socket.
- int [connectToBackend](#) ()
Connects to a backend server.
- void [handleClient](#) (int client_socket)
Handles communication between a client and a backend server.
- void [start](#) ()
Starts the load balancer.

Private Attributes

- int `server_fd`
Server socket file descriptor.
- struct sockaddr_in `address`
Server address structure.
- int `PORT`
Port number the load balancer listens on.
- std::string `server1_ip`
IP address of the first backend server.
- int `server1_port`
Port number of the first backend server.
- std::string `server2_ip`
IP address of the second backend server.
- int `server2_port`
Port number of the second backend server.
- int `current_server`
Counter for round-robin server selection.

Static Private Attributes

- static const int `MAX_CLIENTS` = 2000
Maximum number of simultaneous client connections.

3.3.1 Detailed Description

Implements a round-robin load balancer for multiple backend servers.

This class distributes incoming client connections between multiple backend servers using a round-robin algorithm. It forwards data between clients and backend servers transparently.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 LoadBalancer()

```
LoadBalancer::LoadBalancer (
    int port,
    const std::string & s1_ip,
    int s1_port,
    const std::string & s2_ip,
    int s2_port ) [inline]
```

Constructor.

Parameters

<i>port</i>	Port number for the load balancer
<i>s1_ip</i>	IP address of the first backend server
<i>s1_port</i>	Port number of the first backend server
<i>s2_ip</i>	IP address of the second backend server
<i>s2_port</i>	Port number of the second backend server

Initializes the load balancer with the specified port and backend server details.

3.3.2.2 ~LoadBalancer()

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

Destructor.

Closes the server socket.

3.3.3 Member Function Documentation**3.3.3.1 connectToBackend()**

```
int LoadBalancer::connectToBackend ( ) [inline]
```

Connects to a backend server.

Returns

Socket file descriptor for the backend connection, or -1 on failure

Selects a backend server using round-robin and establishes a connection to it.

3.3.3.2 handleClient()

```
void LoadBalancer::handleClient (
    int client_socket ) [inline]
```

Handles communication between a client and a backend server.

Parameters

<i>client_socket</i>	Socket file descriptor for the client connection
----------------------	--

Establishes a connection to a backend server and forwards data between the client and the selected backend server.

3.3.3.3 setupServer()

```
void LoadBalancer::setupServer ( ) [inline]
```

Sets up the server socket.

Creates and configures the server socket, binds it to the port, and starts listening for connections.

3.3.3.4 start()

```
void LoadBalancer::start ( ) [inline]
```

Starts the load balancer.

Main event loop that accepts client connections and handles them by creating a new process for each connection.

3.3.4 Member Data Documentation

3.3.4.1 address

```
struct sockaddr_in LoadBalancer::address [private]
```

Server address structure.

3.3.4.2 current_server

```
int LoadBalancer::current_server [private]
```

Counter for round-robin server selection.

3.3.4.3 MAX_CLIENTS

```
const int LoadBalancer::MAX_CLIENTS = 2000 [static], [private]
```

Maximum number of simultaneous client connections.

3.3.4.4 PORT

```
int LoadBalancer::PORT [private]
```

Port number the load balancer listens on.

3.3.4.5 server1_ip

```
std::string LoadBalancer::server1_ip [private]
```

IP address of the first backend server.

3.3.4.6 server1_port

```
int LoadBalancer::server1_port [private]
```

Port number of the first backend server.

3.3.4.7 server2_ip

```
std::string LoadBalancer::server2_ip [private]
```

IP address of the second backend server.

3.3.4.8 server2_port

```
int LoadBalancer::server2_port [private]
```

Port number of the second backend server.

3.3.4.9 server_fd

```
int LoadBalancer::server_fd [private]
```

Server socket file descriptor.

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

- [src/load_balancer.cpp](#)

Chapter 4

File Documentation

4.1 src/blink_server.cpp File Reference

Implementation of the [BlinkServer](#) class.

```
#include "blink_server.h"
```

Include dependency graph for blink_server.cpp:

4.2 src/blink_server.h File Reference

Header file for the [BlinkServer](#) class implementing a Redis-like server.

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <sstream>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <poll.h>
#include <fcntl.h>
#include <cstring>
#include <memory>
#include <algorithm>
#include <sys/epoll.h>
#include "blinkdb.h"
```

Include dependency graph for blink_server.h: This graph shows which files directly or indirectly include this file:

Classes

- class [BlinkServer](#)

Implements a Redis-compatible server using the RESP-2 protocol.

Macros

- `#define __BLINK_SERVER_H`

4.2.1 Detailed Description

Header file for the [BlinkServer](#) class implementing a Redis-like server.

Author

Madhumita

Date

2025-03-31

4.2.2 Macro Definition Documentation

4.2.2.1 __BLINK_SERVER_H

```
#define __BLINK_SERVER_H
```

4.3 src/blinkdb.cpp File Reference

Implementation of the [BlinkDB](#) class.

```
#include "blinkdb.h"
```

Include dependency graph for blinkdb.cpp:

4.3.1 Detailed Description

Implementation of the [BlinkDB](#) class.

Author

Madhumita

Date

2025-03-31

4.4 src/blinkdb.h File Reference

Header file for the [BlinkDB](#) in-memory database with LRU caching.

```
#include <unordered_map>
#include <list>
#include <string>
#include <fstream>
#include <mutex>
#include <shared_mutex>
#include <thread>
#include <chrono>
#include <iostream>
#include <future>
#include <exception>
#include <cstdio>
```

Include dependency graph for blinkdb.h: This graph shows which files directly or indirectly include this file:

Classes

- class [BlinkDB](#)
An in-memory key-value database with LRU caching and disk persistence.

Macros

- #define [VALUE_SIZE](#) 256
- #define [MAX_CAPACITY](#) 10000
- #define [FLUSH_FILE](#) "flush_data.txt"
- #define [COMPACTION_THRESHOLD](#) 1000

4.4.1 Detailed Description

Header file for the [BlinkDB](#) in-memory database with LRU caching.

Author

Madhumita

Date

2025-03-31

4.4.2 Macro Definition Documentation

4.4.2.1 COMPACTION_THRESHOLD

```
#define COMPACTION_THRESHOLD 1000
```

4.4.2.2 FLUSH_FILE

```
#define FLUSH_FILE "flush_data.txt"
```

4.4.2.3 MAX_CAPACITY

```
#define MAX_CAPACITY 10000
```

4.4.2.4 VALUE_SIZE

```
#define VALUE_SIZE 256
```

4.5 src/load_balancer.cpp File Reference

Implementation of a load balancer for [BlinkDB](#) servers.

```
#include <iostream>
#include <vector>
#include <string>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <poll.h>
#include <fcntl.h>
```

Include dependency graph for load_balancer.cpp:

Classes

- class [LoadBalancer](#)
Implements a round-robin load balancer for multiple backend servers.

Functions

- int [main](#) (int argc, char *argv[])
Main function.

4.5.1 Detailed Description

Implementation of a load balancer for [BlinkDB](#) servers.

Author

Madhumita

Date

2025-03-31

This file implements a load balancer that distributes client connections between multiple [BlinkDB](#) server instances using a round-robin algorithm.

4.5.2 Function Documentation

4.5.2.1 main()

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

Main function.

Parameters

<i>argc</i>	Number of command-line arguments
<i>argv</i>	Array of command-line arguments

Returns

Exit code (0 for success, 1 for failure)

Parses command-line arguments and starts the load balancer with the specified configuration.

4.6 src/main.cpp File Reference

Main entry point for the [BlinkServer](#) application.

```
#include "blink_server.h"
Include dependency graph for main.cpp:
```

Functions

- `int main ()`
Main function.

4.6.1 Detailed Description

Main entry point for the [BlinkServer](#) application.

Author

Madhumita

Date

2025-03-31

This file contains the main function that initializes and starts the [BlinkServer](#), handling any exceptions that may occur during startup.

4.6.2 Function Documentation

4.6.2.1 `main()`

```
int main ( )
```

Main function.

Returns

Exit code (0 for success, 1 for failure)

Creates and starts a [BlinkServer](#) instance, catching and reporting any exceptions that occur during server startup.

Index

- __BLINK_SERVER_H
 - blink_server.h, [24](#)
- ~BlinkDB
 - BlinkDB, [6](#)
- ~BlinkServer
 - BlinkServer, [12](#)
- ~LoadBalancer
 - LoadBalancer, [20](#)
- address
 - BlinkServer, [17](#)
 - LoadBalancer, [21](#)
- blink_server.h
 - __BLINK_SERVER_H, [24](#)
- BlinkDB, [5](#)
 - ~BlinkDB, [6](#)
 - BlinkDB, [6](#)
 - clearPersistenceFile, [7](#)
 - db_mutex, [9](#)
 - del, [7](#)
 - dirty, [9](#)
 - evicted_keys, [9](#)
 - flushToDiskAsync, [7](#)
 - flushToDiskPeriodically, [7](#)
 - get, [7](#)
 - loadFromFile, [8](#)
 - lru_keys, [9](#)
 - lru_map, [10](#)
 - max_cache_size, [10](#)
 - persistence_file, [10](#)
 - persistToFile, [8](#)
 - restoreFromDisk, [8](#)
 - set, [8](#)
 - store, [10](#)
 - updateLRU, [9](#)
- blinkdb.h
 - COMPACTION_THRESHOLD, [25](#)
 - FLUSH_FILE, [26](#)
 - MAX_CAPACITY, [26](#)
 - VALUE_SIZE, [26](#)
- BlinkServer, [10](#)
 - ~BlinkServer, [12](#)
 - address, [17](#)
 - BlinkServer, [12](#)
 - database, [17](#)
 - decodeCommand, [12](#)
 - encodeBulkString, [13](#)
 - encodeError, [13](#)
 - encodeInteger, [13](#)
 - encodeSimpleString, [14](#)
 - handleClientConnections, [14](#)
 - handleClientRead, [14](#)
 - handleCommand, [15](#)
 - MAX_CLIENTS, [17](#)
 - PORT, [18](#)
 - processDel, [15](#)
 - processGet, [16](#)
 - processSet, [16](#)
 - server_fd, [18](#)
 - setupServer, [17](#)
 - start, [17](#)
- clearPersistenceFile
 - BlinkDB, [7](#)
- COMPACTION_THRESHOLD
 - blinkdb.h, [25](#)
- connectToBackend
 - LoadBalancer, [20](#)
- current_server
 - LoadBalancer, [21](#)
- database
 - BlinkServer, [17](#)
- db_mutex
 - BlinkDB, [9](#)
- decodeCommand
 - BlinkServer, [12](#)
- del
 - BlinkDB, [7](#)
- dirty
 - BlinkDB, [9](#)
- encodeBulkString
 - BlinkServer, [13](#)
- encodeError
 - BlinkServer, [13](#)
- encodeInteger
 - BlinkServer, [13](#)
- encodeSimpleString
 - BlinkServer, [14](#)
- evicted_keys
 - BlinkDB, [9](#)
- FLUSH_FILE
 - blinkdb.h, [26](#)
- flushToDiskAsync
 - BlinkDB, [7](#)
- flushToDiskPeriodically
 - BlinkDB, [7](#)

- get
 - BlinkDB, 7
- handleClient
 - LoadBalancer, 20
- handleClientConnections
 - BlinkServer, 14
- handleClientRead
 - BlinkServer, 14
- handleCommand
 - BlinkServer, 15
- load_balancer.cpp
 - main, 27
- LoadBalancer, 18
 - ~LoadBalancer, 20
 - address, 21
 - connectToBackend, 20
 - current_server, 21
 - handleClient, 20
 - LoadBalancer, 19
 - MAX_CLIENTS, 21
 - PORT, 21
 - server1_ip, 22
 - server1_port, 22
 - server2_ip, 22
 - server2_port, 22
 - server_fd, 22
 - setupServer, 21
 - start, 21
- loadFromFile
 - BlinkDB, 8
- lru_keys
 - BlinkDB, 9
- lru_map
 - BlinkDB, 10
- main
 - load_balancer.cpp, 27
 - main.cpp, 28
- main.cpp
 - main, 28
- max_cache_size
 - BlinkDB, 10
- MAX_CAPACITY
 - blinkdb.h, 26
- MAX_CLIENTS
 - BlinkServer, 17
 - LoadBalancer, 21
- persistence_file
 - BlinkDB, 10
- persistToFile
 - BlinkDB, 8
- PORT
 - BlinkServer, 18
 - LoadBalancer, 21
- processDel
 - BlinkServer, 15
- processGet
 - BlinkServer, 16
- processSet
 - BlinkServer, 16
- restoreFromDisk
 - BlinkDB, 8
- server1_ip
 - LoadBalancer, 22
- server1_port
 - LoadBalancer, 22
- server2_ip
 - LoadBalancer, 22
- server2_port
 - LoadBalancer, 22
- server_fd
 - BlinkServer, 18
 - LoadBalancer, 22
- set
 - BlinkDB, 8
- setupServer
 - BlinkServer, 17
 - LoadBalancer, 21
- src/blink_server.cpp, 23
- src/blink_server.h, 23
- src/blinkdb.cpp, 24
- src/blinkdb.h, 25
- src/load_balancer.cpp, 26
- src/main.cpp, 27
- start
 - BlinkServer, 17
 - LoadBalancer, 21
- store
 - BlinkDB, 10
- updateLRU
 - BlinkDB, 9
- VALUE_SIZE
 - blinkdb.h, 26