BLINK DB 24CS60R02 - PART B

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 BenchmarkData Class Reference

Generates and stores test data for benchmarking.

```
#include <benchmarkdata.h>
```

Public Member Functions

- BenchmarkData (size_t reads=1000000, size_t writes=1000000, size_t keyLength=16, size_t value←Length=16)

  Constructor.

Public Attributes

- std::vector< std::string > **keys**

  Pre-generated keys.
- std::vector< std::string > **values**

  Pre-generated values.

Private Member Functions

- std::string generateRandomString (size_t length)

  Generates a random string of specified length.
- void generateTestData ()

  Generates all test data (keys and values)

Private Attributes

- size_t **numReads**

  Number of read operations to generate data for.
- size_t **numWrites**

  Number of write operations to generate data for.
- size_t **keySize**

  Size of generated keys in characters.
- size_t **valueSize**

  Size of generated values in characters.

3.1.1 Detailed Description

Generates and stores test data for benchmarking.

This class pre-generates random keys and values for consistent benchmark testing of storage engines.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BenchmarkData()

```
BenchmarkData::BenchmarkData (
             size_t reads = 1000000,
             size_t writes = 1000000,
             size_t keyLength = 16,
             size_t valueLength = 16)
```

Constructor.

Constructs a BenchmarkData object with the given parameters.

Parameters

reads	Number of read operations (default: 1,000,000)
writes	Number of write operations (default: 1,000,000)
keyLength	Length of each key (default: 16)
valueLength	Length of each value (default: 16)

Initializes the number of read and write operations, key and value sizes, and generates the necessary test data.

Parameters

reads	Number of read operations.
writes	Number of write operations.
keyLength	Length of each key.
valueLength	Length of each value.

3.1.3 Member Function Documentation

3.1.3.1 generateRandomString()

```
std::string BenchmarkData::generateRandomString (
             size_t length)  [private]
```

Generates a random string of specified length.

Generates a random alphanumeric string of a specified length.

Parameters

length	The length of the string to generate

Returns

A random string

Uses a random number generator to create a string consisting of uppercase letters, lowercase letters, and digits.

Parameters

length	Length of the string to generate.

Returns

A randomly generated string of the specified length.

3.1.3.2 generateTestData()

```
void BenchmarkData::generateTestData ()  [private]
```

Generates all test data (keys and values)

Generates test data consisting of random keys and values.

Populates the `keys` vector with randomly generated keys and the `values` vector with randomly generated values.

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

- benchmarkdata/benchmarkdata.h
- benchmarkdata/benchmarkdata.cpp

3.2 KQueueServer Class Reference

Server implementation using kqueue for I/O multiplexing.

```
#include <server.h>
```

Public Member Functions

- KQueueServer (LSMTree &store, BenchmarkData &data)

  Constructor.
- ∼KQueueServer ()

  Destructor.
- int run ()

  Initializes and starts the server.

Private Member Functions

- std::string processCommand (const std::vector< std::string > &args, int rn)

  Processes a command from a client.
- void handleClient (int fd, int rn)

  Handles a client connection.

Private Attributes

- int **server_fd**
- int **kq**
- LSMTree & **store**
- BenchmarkData & **data**

3.2.1 Detailed Description

Server implementation using kqueue for I/O multiplexing.

Handles client connections and processes Redis-compatible commands using the LSM Tree storage engine.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 KQueueServer()

```
KQueueServer::KQueueServer (
            LSMTree & store,
            BenchmarkData & data)
```

Constructor.

Constructs a KQueueServer object.

Parameters

store	LSM Tree storage engine
data	Benchmark data
store	Reference to the LSMTree storage engine.
data	Reference to the BenchmarkData generator.

3.2.2.2 ∼KQueueServer()

```
KQueueServer::∼KQueueServer ()
```

Destructor.

Destroys the KQueueServer object and closes open file descriptors.

3.2.3 Member Function Documentation

3.2.3.1 handleClient()

```
void KQueueServer::handleClient (
            int fd,
            int rn) [private]
```

Handles a client connection.

Handles client requests.

Parameters

fd	Client socket file descriptor
rn	Random number for benchmark data access
fd	Client socket file descriptor.
rn	Random index for benchmark data access.

3.2.3.2 processCommand()

```
std::string KQueueServer::processCommand (
            const std::vector< std::string > & args,
            int rn) [private]
```

Processes a command from a client.

Processes a client command and generates a response.

Parameters

args	Command arguments
rn	Random number for benchmark data access

Returns

Response to send to client

Parameters

args	Parsed command arguments.
rn	Random index for benchmark data access.

Returns

A RESP-formatted response string.

3.2.3.3 run()

```
int KQueueServer::run ()
```

Initializes and starts the server.

Runs the KQueue-based event-driven server.

Returns

0 on success, error code otherwise

0 on successful execution, 1 on error.

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

- server/server.h
- server/server.cpp

3.3 RespParser Class Reference

Parser for Redis RESP-2 protocol.

```
#include <resp_parser.h>
```

Static Public Member Functions

- static std::vector< std::string > parseArray (const std::string &buffer)

  Parses a RESP array message into command arguments.
- static std::string serializeBulkString (const std::string &value)

  Serializes a string into RESP bulk string format.
- static std::string createSimpleString (const std::string &status)

  Creates a RESP simple string (status) response.
- static std::string createError (const std::string &error)

  Creates a RESP error response.

3.3.1 Detailed Description

Parser for Redis RESP-2 protocol.

Handles serialization and deserialization of Redis RESP-2 protocol messages.

3.3.2 Member Function Documentation

3.3.2.1 createError()

```
std::string RespParser::createError (
                const std::string & error)   [static]
```

Creates a RESP error response.

Creates a RESP-2 error response.

Parameters

error	The error message

Returns

RESP formatted error message

Error messages start with '-' and contain an error description.

Parameters

error	The error message.

Returns

A RESP-2 formatted error response.

3.3.2.2 createSimpleString()

```
std::string RespParser::createSimpleString (
             const std::string & status)  [static]
```

Creates a RESP simple string (status) response.

Creates a RESP-2 simple string response.

Parameters

status	The status message

Returns

RESP formatted simple string

Simple strings are used for success messages and start with '+'.

Parameters

status	The success message to return.

Returns

A RESP-2 formatted simple string.

3.3.2.3 parseArray()

```
std::vector< std::string > RespParser::parseArray (
            const std::string & buffer)  [static]
```

Parses a RESP array message into command arguments.

Parses a RESP-2 array message and extracts command arguments.

Parameters

buffer	The RESP message buffer

Returns

Vector of command arguments

RESP-2 arrays start with '*' followed by the number of elements, and each element is a bulk string prefixed with '$' followed by its length.

Parameters

buffer	The RESP-2 formatted string received from the client.

Returns

A vector containing the parsed arguments as strings.

3.3.2.4 serializeBulkString()

```
std::string RespParser::serializeBulkString (
            const std::string & value)  [static]
```

Serializes a string into RESP bulk string format.

Serializes a string into a RESP-2 bulk string format.

Parameters

value	The string to serialize

Returns

RESP formatted bulk string

If the string is empty, it returns the RESP-2 null bulk string "$-1\r\n".

Parameters

value	The string to be serialized.

Returns

A RESP-2 formatted bulk string.

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

- server/resp_parser.h
- server/resp_parser.cpp

Chapter 4

File Documentation

4.1 benchmarkdata/benchmarkdata.cpp File Reference

Implementation of benchmark data generator.

```
#include "benchmarkdata.h"
#include <iostream>
#include <random>
#include <algorithm>
```

4.1.1 Detailed Description

Implementation of benchmark data generator.

4.2 benchmarkdata/benchmarkdata.h File Reference

Benchmark data generator for LSM Tree testing.

```
#include <string>
#include <vector>
```

Classes

- class BenchmarkData

  Generates and stores test data for benchmarking.

4.2.1 Detailed Description

Benchmark data generator for LSM Tree testing.

4.3 benchmarkdata.h

Go to the documentation of this file.

```


#ifndef BENCHMARK_DATA_H
#define BENCHMARK_DATA_H

#include <string>
#include <vector>

class BenchmarkData
{
private:
    size_t numReads;
    size_t numWrites;
    size_t keySize;
    size_t valueSize;

    std::string generateRandomString(size_t length);

    void generateTestData();

public:
    std::vector<std::string> keys;
    std::vector<std::string> values;

    BenchmarkData(
        size_t reads = 1000000,
        size_t writes = 1000000,
        size_t keyLength = 16,
        size_t valueLength = 16);
};

#endif // BENCHMARK_DATA_H
```

4.4 main.cpp File Reference

Main entry point for the BLINK DB server application.

```
#include "server/server.h"
#include "benchmarkdata/benchmarkdata.h"
#include "../../part_a/src/StorageEngine/lsmtree.h"
#include <iostream>
#include <string>
#include <stdexcept>
```

Functions

- int **main** ()

4.4.1 Detailed Description

Main entry point for the BLINK DB server application.

4.5 server/resp_parser.cpp File Reference

Implementation of Redis RESP-2 protocol parser.

```
#include "resp_parser.h"
#include <sstream>
```

4.5.1 Detailed Description

Implementation of Redis RESP-2 protocol parser.

4.6 server/resp_parser.h File Reference

Redis RESP-2 protocol parser implementation header file.

```
#include <string>
#include <vector>
```

Classes

- class RespParser

  Parser for Redis RESP-2 protocol.

4.6.1 Detailed Description

Redis RESP-2 protocol parser implementation header file.

4.7 resp_parser.h

Go to the documentation of this file.

```


#ifndef RESP_PARSER_H
#define RESP_PARSER_H

#include <string>
#include <vector>

class RespParser
{
public:
    static std::vector<std::string> parseArray(const std::string &buffer);

    static std::string serializeBulkString(const std::string &value);

    static std::string createSimpleString(const std::string &status);

    static std::string createError(const std::string &error);
};

#endif // RESP_PARSER_H
```

4.8 server/server.cpp File Reference

Implementation of KQueue-based server for handling key-value store operations.

```
#include "server.h"
#include "resp_parser.h"
#include <iostream>
#include <sys/event.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <algorithm>
#include <random>
```

4.8.1 Detailed Description

Implementation of KQueue-based server for handling key-value store operations.

4.9 server/server.h File Reference

KQueue-based server for LSM Tree storage engine.

```
#include <string>
#include <vector>
#include "../../part_a/src/StorageEngine/lsmtree.h"
#include "../benchmarkdata/benchmarkdata.h"
```

Classes

- class KQueueServer

  Server implementation using kqueue for I/O multiplexing.

Macros

- #define **PORT** 9002
- #define **MAX_EVENTS** 1024
- #define **BUFFER_SIZE** 1024

4.9.1 Detailed Description

KQueue-based server for LSM Tree storage engine.

4.10 server.h

Go to the documentation of this file.

```

#ifndef SERVER_H
#define SERVER_H

#include <string>
#include <vector>
#include "../../part_a/src/StorageEngine/lsmtree.h"
#include "../benchmarkdata/benchmarkdata.h"

#define PORT 9002
#define MAX_EVENTS 1024
#define BUFFER_SIZE 1024

class KQueueServer
{
private:
    int server_fd;
    int kq;
    LSMTree &store;
    BenchmarkData &data;

    std::string processCommand(const std::vector<std::string> &args, int rn);

    void handleClient(int fd, int rn);

public:
    KQueueServer(LSMTree &store, BenchmarkData &data);

    ~KQueueServer();

    int run();
};

#endif // SERVER_H
```

Index