Part 3

Generated by Doxygen 1.9.8

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 ClientContext Struct Reference

Maintains the context and state of each connected client.

Public Types

- enum { **PARSE_TYPE** , **PARSE_ARGUMENTS** }

Public Member Functions

- ClientContext (int fd_)

  Constructor to initialize the client context with a file descriptor.

Public Attributes

- int **fd**

  File descriptor for the client connection.
- std::string **buffer**

  Data buffer for storing incoming data from the client.
- std::vector< std::string > **args**

  List of arguments for the current command.
- size_t **expected_bulk_length**

  Expected length of the next bulk string argument.
- int **expected_args**

  Number of arguments expected for the current command.
- enum ClientContext:: { ... } **state**

  Current state of the parsing process.

3.1.1 Detailed Description

Maintains the context and state of each connected client.

This structure is used to track the client's socket file descriptor (`fd`), the buffer of data being parsed, and the parsing state. It also tracks the arguments of the current command being processed.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ClientContext()

```
ClientContext::ClientContext (
            int fd_ )  [inline]
```

Constructor to initialize the client context with a file descriptor.

Parameters

*fd_*	The file descriptor for the client connection

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

- server.cpp

3.2 ThreadPool Class Reference

A simple thread pool to handle client connections concurrently.

Public Member Functions

- ThreadPool (size_t numThreads)
  Constructs a thread pool with a specified number of threads.
- ~**ThreadPool** ()
  Destructor to join all threads and clean up resources.
- void enqueue (std::function< void()> task)
  Enqueues a new task to be executed by a worker thread.

3.2.1 Detailed Description

A simple thread pool to handle client connections concurrently.

The thread pool processes tasks such as handling client requests, parsing commands, and interacting with the REPL data store. Tasks are queued and executed by worker threads.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ThreadPool()

```
ThreadPool::ThreadPool (
            size_t numThreads )
```

Constructs a thread pool with a specified number of threads.

Parameters

numThreads	The number of worker threads in the pool

3.2.3 Member Function Documentation

3.2.3.1 enqueue()

```
void ThreadPool::enqueue (
            std::function< void()> task )
```

Enqueues a new task to be executed by a worker thread.

Parameters

task	A function to be executed by a worker thread

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

- server.cpp

Chapter 4

File Documentation

4.1 client.cpp File Reference

A simple client that communicates with a server using RESP (REdis Serialization Protocol). This client sends commands to the server and processes the server's response.

```
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
```

Include dependency graph for client.cpp:

4.2 server.cpp File Reference

A simple multi-threaded server that processes client requests using the RESP (REdis Serialization Protocol). The server handles SET and GET commands, stores data, and uses `epoll` for scalable I/O handling.

```
#include <iostream>
#include <string>
#include <unordered_map>
#include <vector>
#include <cstring>
#include <unistd.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <sys/epoll.h>
#include <thread>
#include <mutex>
#include <queue>
#include <atomic>
#include <condition_variable>
#include "REPL.h"
```

Include dependency graph for server.cpp:

Classes

- struct ClientContext

  Maintains the context and state of each connected client.
- class ThreadPool

  A simple thread pool to handle client connections concurrently.

Macros

- #define **MAX_EVENTS** 1024

  Maximum number of events that can be handled at once by epoll.
- #define **BUFFER_SIZE** 4096

  Buffer size for receiving data from the client.
- #define **PORT** 6379

  Port number on which the server listens.

Functions

- void set_nonblocking (int fd)

  Sets a socket to non-blocking mode.
- void send_response (int fd, const std::string &response)

  Sends a response to the client.
- void handle_client (ClientContext &ctx)

  Handles the client's requests, parsing commands and interacting with the REPL data store. This function is responsible for parsing the RESP commands from the client, executing commands like "SET" and "GET", and sending responses back to the client.
- void SIGHANDLER (int sig)

  Signal handler to cleanly shut down the server on SIGINT (Ctrl+C). This function cleans up resources when the server is interrupted and exits the program.
- int main ()

  The main entry point for the server application. This function sets up the server socket, listens for incoming client connections, and uses `epoll` to handle multiple clients concurrently. It also processes client commands using the thread pool for parallel execution.

Variables

- atomic_flag **flag** = ATOMIC_FLAG_INIT

  Atomic flag used for synchronization between threads.
- REPL **r**

  Global REPL object for storing key-value pairs in memory.

4.2.1 Detailed Description

A simple multi-threaded server that processes client requests using the RESP (REdis Serialization Protocol). The server handles SET and GET commands, stores data, and uses `epoll` for scalable I/O handling.

4.2.2 Function Documentation

4.2.2.1 handle_client()

```
void handle_client (
            ClientContext & ctx )
```

Handles the client's requests, parsing commands and interacting with the REPL data store. This function is responsible for parsing the RESP commands from the client, executing commands like "SET" and "GET", and sending responses back to the client.

Parameters

ctx	The client context containing the state and data for the client

4.2.2.2 main()

```
int main ( )
```

The main entry point for the server application. This function sets up the server socket, listens for incoming client connections, and uses `epoll` to handle multiple clients concurrently. It also processes client commands using the thread pool for parallel execution.

Returns

0 on success, -1 on failure

4.2.2.3 send_response()

```
void send_response (
            int fd,
            const std::string & response )
```

Sends a response to the client.

Parameters

fd	The file descriptor of the client socket
response	The response message to send to the client

4.2.2.4 set_nonblocking()

```
void set_nonblocking (
            int fd )
```

Sets a socket to non-blocking mode.

Parameters

fd	The file descriptor of the socket

4.2.2.5 SIGHANDLER()

```
void SIGHANDLER (
            int sig )
```

Signal handler to cleanly shut down the server on SIGINT (Ctrl+C). This function cleans up resources when the server is interrupted and exits the program.

Parameters

sig	The signal number (ignored in this implementation)

Index