BlinkDB Documentation

Generated by Doxygen 1.9.1

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 files with brief descriptions:

Chapter 3

Class Documentation

3.1 BloomFilter Class Reference

Implements a simple Bloom filter for fast key existence checks.

```
#include <bloomfilter.h>
```

Public Member Functions

- BloomFilter (int _size=10000)

  Constructor to initialize the Bloom filter with a given size.
- void insert (const std::string &_key)

  Inserts a key into the Bloom filter.
- bool contains (const std::string &_key)

  Checks if a key exists in the Bloom filter.
- void remove (const std::string &_key)

  Removes a key from the Bloom filter (Note: Bloom filters generally do not support removals correctly).

3.1.1 Detailed Description

Implements a simple Bloom filter for fast key existence checks.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BloomFilter()

```
BloomFilter::BloomFilter (
            int _size = 10000 )  [explicit]
```

Constructor to initialize the Bloom filter with a given size.

Parameters

_size	The size of the Bloom filter (default: 10,000).

```
: filter(_size, false), filter_size(_size) {}
```

3.1.3 Member Function Documentation

3.1.3.1 contains()

```
bool BloomFilter::contains (
             const std::string & _key )
```

Checks if a key exists in the Bloom filter.

Parameters

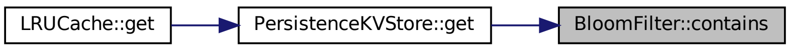
_key	The key to check.

Returns

True if the key is possibly in the filter, false otherwise.

```
{
    return filter[hashKey(_key)];
}
```

Here is the caller graph for this function:



$$\begin{tikzcd} \text{LRUCache::get} \arrow[r] & \text{PersistenceKVStore::get} \arrow[r] & \text{BloomFilter::contains} \end{tikzcd}$$

3.1.3.2 insert()

```
void BloomFilter::insert (
             const std::string & _key )
```

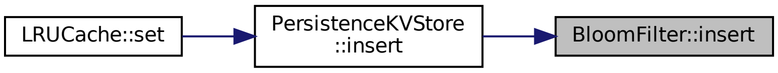
Inserts a key into the Bloom filter.

Parameters

_key	The key to insert.

```
{
    filter[hashKey(_key)] = true;
}
```

Here is the caller graph for this function:



3.1.3.3 remove()

```
void BloomFilter::remove (
            const std::string & _key )
```

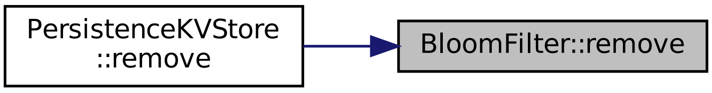
Removes a key from the Bloom filter (Note: Bloom filters generally do not support removals correctly).

Parameters

_key	The key to remove.

```
{
    filter[hashKey(_key)] = false;
}
```

Here is the caller graph for this function:



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

- lib/bloomfilter.h

3.2 Client Class Reference

A class to interact with a Blink server using RESP (Blink Serialization Protocol).

`#include <client.h>`

Public Member Functions

- Client (std::string _ip_addr, int _port, int _buffer_size)

  Constructor for Client class.
- ∼Client ()

  Destructor to free allocated memory.
- int server_init ()

  Initializes the connection to the Blink server.
- std::string set (const std::string &_key, const std::string &_value)

  Sends a SET command to store a key-value pair.
- std::string get (const std::string &_key)

  Sends a GET command to retrieve the value of a key.
- std::string del (const std::string &_key)

  Sends a DEL command to delete a key.
- void close_server ()

  Closes the connection to the Blink server.

Public Attributes

- int buffer_size

  Buffer size for reading responses.
- std::string ip_addr

  IP address of the Blink server.
- char ∗ buffer = nullptr

  Dynamic buffer for receiving data.
- int port

  Port number of the Blink server.

3.2.1 Detailed Description

A class to interact with a Blink server using RESP (Blink Serialization Protocol).

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Client()

```
Client::Client (
             std::string _ip_addr,
             int _port,
             int _buffer_size )  [inline]
```

Constructor for Client class.

Parameters

_ip_addr	IP address of the Blink server.
_port	Port number of the Blink server.
_buffer_size	Size of the buffer for reading responses.

```
        : ip_addr(_ip_addr)
    {
        port   = _port;
        buffer_size = _buffer_size;
        buffer = new char[buffer_size];
    }
```

3.2.2.2 ∼Client()

`Client::∼Client ( )  [inline]`

Destructor to free allocated memory.

```
    {
        delete[] buffer;
    }
```

3.2.3 Member Function Documentation

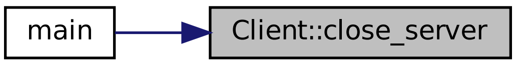
3.2.3.1 close_server()

`void Client::close_server ( )  [inline]`

Closes the connection to the Blink server.

```
    {
        close(sock);
    }
```

Here is the caller graph for this function:



3.2.3.2 del()

```
std::string Client::del (
             const std::string & _key )  [inline]
```

Sends a DEL command to delete a key.

Parameters

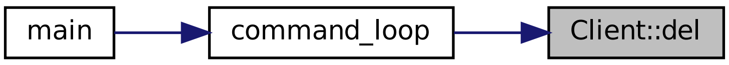
_key	The key.

Returns

Response from the Blink server.

```
{
    return decode_resp(send_req(encode_command("DEL " + _key)));
}
```

Here is the caller graph for this function:



3.2.3.3 get()

```
std::string Client::get (
            const std::string & _key )  [inline]
```

Sends a GET command to retrieve the value of a key.

Parameters

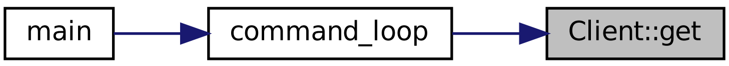
_key	The key.

Returns

Response from the Blink server.

```
{
    return decode_resp(send_req(encode_command("GET " + _key)));
}
```

Here is the caller graph for this function:



3.2.3.4 server_init()

```
int Client::server_init ( )  [inline]
```

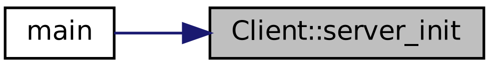
Initializes the connection to the Blink server.

Returns

Socket descriptor or -1 on failure.

```
{
    sock = 0;
    struct sockaddr_in serv_addr;

    // Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        std::cerr « "Socket creation error" « std::endl;
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);

    // Convert IPv4 address from text to binary
    if (inet_pton(AF_INET, ip_addr.c_str(), &serv_addr.sin_addr) <= 0)
    {
        std::cerr « "Invalid address / Address not supported" « std::endl;
        return -1;
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        std::cerr « "Connection failed" « std::endl;
        return -1;
    }

    return 1;
}
```

Here is the caller graph for this function:



3.2.3.5 set()

```
std::string Client::set (
             const std::string & _key,
             const std::string & _value )  [inline]
```

Sends a SET command to store a key-value pair.

Parameters

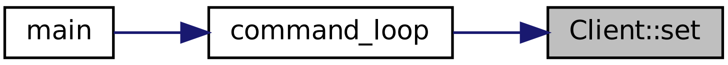
_key	The key.
_value	The value.

Returns

Response from the Blink server.

```
{
    return decode_resp(send_req(encode_command("SET " + _key + " " + _value)));
}
```

Here is the caller graph for this function:



3.2.4 Member Data Documentation

3.2.4.1 buffer

```
char* Client::buffer = nullptr
```

Dynamic buffer for receiving data.

3.2.4.2 buffer_size

```
int Client::buffer_size
```

Buffer size for reading responses.

3.2.4.3 ip_addr

```
std::string Client::ip_addr
```

IP address of the Blink server.

3.2.4.4 port

```
int Client::port
```

Port number of the Blink server.

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

- lib/client.h

3.3 Dict Class Reference

A dictionary (hash table) implementation with dynamic resizing and rehashing.

```
#include <dict.h>
```

Public Member Functions

- Dict (std::function< unsigned int(const void ∗)> hashFunc, std::function< void ∗(const void ∗)> keyDup↩ Func, std::function< void ∗(const void ∗)> valDupFunc, std::function< int(const void ∗, const void ∗)> key↩ CompareFunc, std::function< void(void ∗)> keyDestructorFunc, std::function< void(void ∗)> valDestructor↩ Func)

  Constructor for the dictionary.
- ∼Dict ()

  Destructor for the dictionary.
- void enableResize (bool enable)

  Enables or disables automatic resizing.
- int add (void ∗key, void ∗val)

  Adds a key-value pair to the dictionary with automatic rehashing.
- int replace (void ∗key, void ∗val)

  Replaces a key's value in the dictionary.
- int remove (const void ∗key)

  Removes a key from the dictionary with incremental rehashing.
- void ∗ find (const void ∗key)

  Finds a key in the dictionary.
- int rehash (int n)

  Performs a rehash operation.
- bool isRehashing ()

  Checks if rehashing is in progress.
- size_t get_size_of_dict ()

  Retrieves the total memory usage of the dictionary for keys, values.
- int size ()

  Retrieves the total no of keys in the dictionary.

3.3.1 Detailed Description

A dictionary (hash table) implementation with dynamic resizing and rehashing.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Dict()

```
Dict::Dict (
            std::function< unsigned int(const void *)> hashFunc,
            std::function< void *(const void *)> keyDupFunc,
            std::function< void *(const void *)> valDupFunc,
            std::function< int(const void *, const void *)> keyCompareFunc,
            std::function< void(void *)> keyDestructorFunc,
            std::function< void(void *)> valDestructorFunc )  [inline]
```

Constructor for the dictionary.

Parameters

hashFunc	Hash function.
keyDupFunc	Key duplication function.
valDupFunc	Value duplication function.
keyCompareFunc	Key comparison function.
keyDestructorFunc	Key destructor function.
valDestructorFunc	Value destructor function.

```
        : hashFunction(hashFunc), keyDup(keyDupFunc), valDup(valDupFunc),
          keyCompare(keyCompareFunc), keyDestructor(keyDestructorFunc),
          valDestructor(valDestructorFunc)
    {
        _dictInit(&d);
    }
```

3.3.2.2 ∼Dict()

```
Dict::∼Dict ( )  [inline]
```

Destructor for the dictionary.

```
    {
        _dictClear(&d);
    }
```

3.3.3 Member Function Documentation

3.3.3.1 add()

```
int Dict::add (
            void * key,
            void * val )  [inline]
```

Adds a key-value pair to the dictionary with automatic rehashing.

Parameters

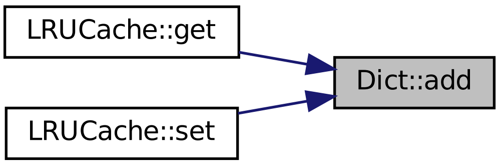
key	Key pointer.
val	Value pointer.

Returns

0 on success, 1 on failure.

```
    {
        // Check if resize is needed before adding
        if (_dictShouldResize())
        {
            // Calculate new size based on current usage
            unsigned long newSize = d.ht[0].used * 2;
            _dictExpand(&d, newSize);
        }

        // Perform a rehash step if rehashing is in progress
        if (dictIsRehashing(&d))
            _dictRehashStep();

        return dictAdd(&d, key, val);
    }
```

Here is the caller graph for this function:



3.3.3.2 enableResize()

```
void Dict::enableResize (
              bool enable )  [inline]
```

Enables or disables automatic resizing.

Parameters

enable	True to enable resizing, false to disable.

```
    {
        dict_can_resize = enable;
    }
```

3.3.3.3 find()

```
void* Dict::find (
            const void * key )  [inline]
```

Finds a key in the dictionary.

Parameters

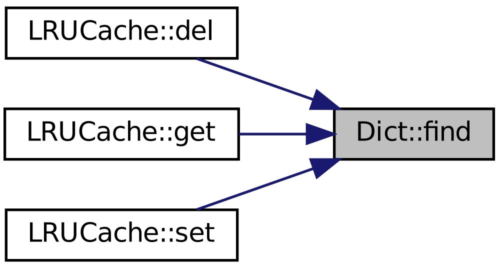
key	Key pointer.

Returns

Pointer to the value if found, nullptr otherwise.

```
    {
        // Perform a rehash step if rehashing is in progress
        if (dictIsRehashing(&d))
            _dictRehashStep();

        dictEntry *he = dictFind(&d, key);
        return he ? he->v.val : nullptr;
    }
```

Here is the caller graph for this function:



3.3.3.4 get_size_of_dict()

```
size_t Dict::get_size_of_dict ( )  [inline]
```

Retrieves the total memory usage of the dictionary for keys, values.

Returns

size_t The total size of the dictionary in bytes.

```
    {
        return total_size_of_dict;
    }
```

3.3.3.5 isRehashing()

```
bool Dict::isRehashing ( )  [inline]
```

Checks if rehashing is in progress.

Returns

True if rehashing, false otherwise.

```
    {
        return dictIsRehashing(&d);
    }
```

3.3.3.6 rehash()

```
int Dict::rehash (
             int n )  [inline]
```

Performs a rehash operation.

Parameters

n	Number of steps to rehash.

Returns

0 on completion, 1 if rehashing is ongoing.

```
    {
        return dictRehash(&d, n);
    }
```

3.3.3.7 remove()

```
int Dict::remove (
             const void * key )  [inline]
```

Removes a key from the dictionary with incremental rehashing.

Parameters

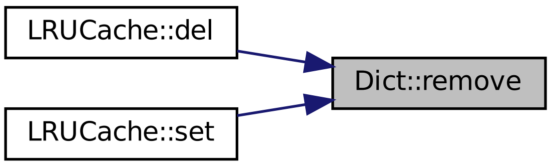
key	Key pointer.

Returns

0 on success, 1 if key not found.

```
    {
        // Perform a rehash step if rehashing is in progress
        if (dictIsRehashing(&d))
            _dictRehashStep();

        return dictDelete(&d, key);
    }
```

Here is the caller graph for this function:



3.3.3.8 replace()

```
int Dict::replace (
              void * key,
              void * val )  [inline]
```

Replaces a key's value in the dictionary.

Parameters

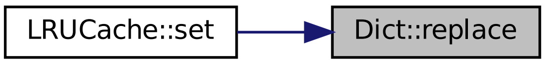
key	Key pointer.
val	Value pointer.

Returns

0 if key already exists and value is replaced, 1 if key is newly added.

```
    {
        return dictReplace(&d, key, val);
    }
```

Here is the caller graph for this function:



3.3.3.9 size()

```
int Dict::size ( ) [inline]
```

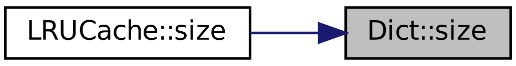
Retrieves the total no of keys in the dictionary.

Returns

int To total no of keys in the dict

```
{
    return d.ht[0].used + d.ht[1].used;
}
```

Here is the caller graph for this function:



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

- lib/dict.h

3.4 LoadBalancer Class Reference

Handles load balancing by distributing client requests to backend servers.

```
#include <load_balancer.h>
```

Public Member Functions

- LoadBalancer (std::string lb_ip, int lb_port, std::vector< ServerAdd > &servers_add, int buffer_size, int max_events)

  Constructs a LoadBalancer instance.
- ∼LoadBalancer ()

  Destructor to clean up resources.
- void server_init (parsingKeyFuncPtr func)

  Initializes the server and starts handling client requests.

3.4.1 Detailed Description

Handles load balancing by distributing client requests to backend servers.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 LoadBalancer()

```
LoadBalancer::LoadBalancer (
               std::string lb_ip,
               int lb_port,
               std::vector< ServerAdd > & servers_add,
               int buffer_size,
               int max_events )
```

Constructs a LoadBalancer instance.

Parameters

lb_port	The port the load balancer listens on.
serves_add	The list of server addresses and ports.
buffer_size	The size of the buffer for client messages.
max_events	The maximum number of events handled by epoll.

```

                        : servers_add_(servers_add)
{
    lb_ip_ = lb_ip;
    lb_port_ = lb_port;
    buffer_size_= buffer_size;
    max_events_= max_events;

    buffer_ = new char[buffer_size];
    for (auto &server_add : servers_add)
    {
        int key = hashKey(server_add.ip + std::to_string(server_add.port));
        hash_ring_.insert(key);
        server_map_[key] = server_add;
    }
}
```

3.4.2.2 ∼LoadBalancer()

```
LoadBalancer::∼LoadBalancer ( )
```

Destructor to clean up resources.

```
{
    delete[] buffer_;
}
```

3.4.3 Member Function Documentation

3.4.3.1 server_init()

```
void LoadBalancer::server_init (
               parsingKeyFuncPtr func )
```

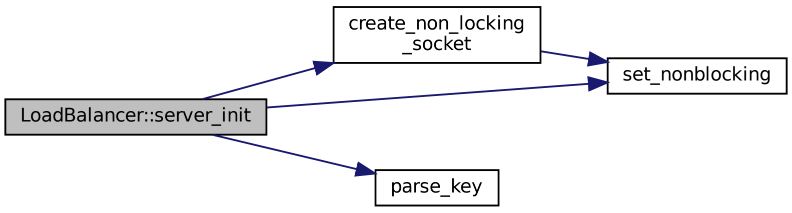
Initializes the server and starts handling client requests.

Parameters

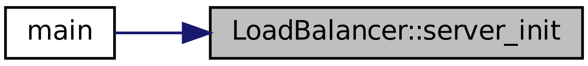
func	Function pointer for parsing the key from client messages.

```
{
    struct sockaddr_in address;
    int lb_sockfd, epoll_fd;
    struct epoll_event event, events[max_events_];
    int addrlen = sizeof(address);

    lb_sockfd = create_non_locking_socket(lb_ip_, lb_port_, address);

    epoll_fd = epoll_create1(0);
    if (epoll_fd == -1)
    {
        perror("[LB]: Epoll creation failed");
        exit(EXIT_FAILURE);
    }

    event.events = EPOLLIN;
    event.data.fd = lb_sockfd;

    if (epoll_ctl(epoll_fd, EPOLL_CTL_ADD, lb_sockfd, &event) == -1)
    {
        perror("[LB]: Epoll_ctl failed");
        exit(EXIT_FAILURE);
    }

    std::cout « "[LB]: Load Balancer listening on port " « lb_port_ « std::endl;

    while (true)
    {
        int num_events = epoll_wait(epoll_fd, events, max_events_, -1);
        if (num_events == -1)
        {
            perror("[LB]: Epoll wait failed");
            break;
        }

        for (int i = 0; i < num_events; ++i)
        {
            int sock_fd = events[i].data.fd;

            if (sock_fd == lb_sockfd)
            {
                // New client connection
                int client_fd = accept(lb_sockfd, (struct sockaddr *)&address, (socklen_t *)&addrlen);

                // char client_ip[INET_ADDRSTRLEN];
                // inet_ntop(AF_INET, &address.sin_addr, client_ip, INET_ADDRSTRLEN);
                // int client_port = ntohs(address.sin_port);

                if (client_fd == -1)
                {
                    perror("[LB]: Accept failed");
                    continue;
                }

                set_nonblocking(client_fd);

                event.events = EPOLLIN | EPOLLET;
                event.data.fd = client_fd;

                if (epoll_ctl(epoll_fd, EPOLL_CTL_ADD, client_fd, &event) == -1)
                {
                    perror("Epoll_ctl client add failed");
                    close(client_fd);
                    continue;
                }

            //    std::cout « "[LB]: New client connected: " « client_ip « ":" « client_port «
    std::endl;
            }
            else
            {
                // Handle client request
                memset(buffer_, 0, buffer_size_);
                int bytes_read = recv(sock_fd, buffer_, buffer_size_, 0);

                if (bytes_read > 0)
                {
                    buffer_[bytes_read] = '\0';
                    std::string key = parse_key(buffer_, bytes_read);
                    ServerAdd server_add = getServer(key);

                    // Create a new connection to the backend server
                    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
                    struct sockaddr_in server_addr;
                    server_addr.sin_family = AF_INET;
                    server_addr.sin_port = htons(server_add.port);

                    if (inet_pton(AF_INET, server_add.ip.c_str(), &server_addr.sin_addr) <= 0)
                    {
                        perror("[LB]: Invalid IP address");
                        exit(EXIT_FAILURE);
                    }

                    if (connect(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) == 0)
                    {
                        send(server_fd, buffer_, bytes_read, 0);
                        // char response[buffer_size_] = {0};
                        int resp_bytes = recv(server_fd, buffer_, buffer_size_, 0);
                        if (resp_bytes > 0)
                        {
                            send(sock_fd, buffer_, resp_bytes, 0);
                        }
                        close(server_fd); // Close backend server connection after response
                    }
                    else
                    {
                        perror("[LB]:erver connection failed");
                        close(server_fd);
                    }
                }
                else
                {
                    // Client disconnected
                    // std::cout « "[LB]: Client " « sock_fd « " disconnected." « std::endl;
                    epoll_ctl(epoll_fd, EPOLL_CTL_DEL, sock_fd, nullptr);
                    close(sock_fd);
                }
            }
        }
    }

    close(lb_sockfd);
    close(epoll_fd);
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



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

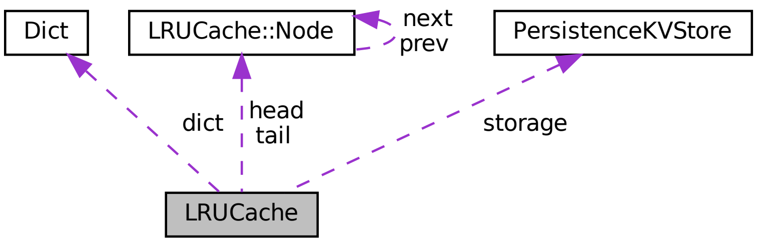
- lib/load_balancer.h

3.5 LRUCache Class Reference

Implements a Least Recently Used (LRU) cache with memory constraints.

```
#include <lru_cache_v0.h>
```

Collaboration diagram for LRUCache:



Classes

- struct Node

  Represents a node in the doubly linked list for the LRU cache.

Public Member Functions

- LRUCache (size_t max_mem)
- void set (const std::string &key, const std::string &value)
- bool get (const std::string &key, std::string &value)
- bool del (const std::string &key)
- size_t memory_usage () const
- size_t max_memory () const
- size_t size () const
- LRUCache (size_t max_mem)

  Constructs a new LRU Cache with the specified memory limit.
- ∼LRUCache ()

  Destroys the LRU Cache and frees all allocated memory.
- std::string get (const void ∗key)

  Retrieves the value for a given key.
- void printList ()

  Prints the current state of the cache for debugging.
- void set (void ∗key, void ∗value)

  Adds or updates a key-value pair in the cache.
- int del (const void ∗key)

Deletes a key-value pair from the cache.
- size_t memory_usage ()

  Gets the current memory usage of the cache.
- size_t max_memory ()

  Gets the maximum memory limit of the cache.
- size_t size ()

  Gets the number of items in the cache.

Public Attributes

- Dict dict
- Node ∗ head
- Node ∗ tail
- PersistenceKVStore storage
- std::string value

3.5.1 Detailed Description

Implements a Least Recently Used (LRU) cache with memory constraints.

This class provides a memory-constrained LRU cache implementation that evicts least recently used items when memory limits are exceeded. It uses a doubly linked list for tracking usage order and a dictionary for O(1) lookups.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 LRUCache() [1/2]

```
LRUCache::LRUCache (
            size_t max_mem ) [inline]
: current_memory_usage(0), max_memory_bytes(max_mem) {}
```

3.5.2.2 LRUCache() [2/2]

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

Constructs a new LRU Cache with the specified memory limit.

Parameters

max_mem	Maximum memory limit in bytes

```
                              :dict(stringHash, nullptr, nullptr, stringCompare, freeKey, freeValue),
      current_memory_usage(0), storage("./blink")
```

```
    {
        max_memory_bytes = max_mem;
        head = new Node(strdup("-1"), strdup("-1"));
        tail = new Node(strdup("-1"), strdup("-1"));
        head->next = tail;
        tail->prev = head;
    }
```

3.5.2.3 ∼LRUCache()

`LRUCache::∼LRUCache ( ) [inline]`

Destroys the LRU Cache and frees all allocated memory.

```
    {
        freeNode(head);
        freeNode(tail);
    }
```

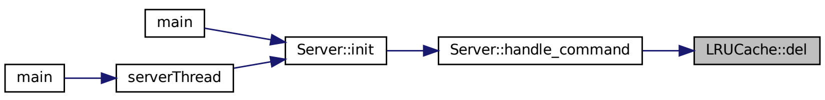
3.5.3 Member Function Documentation

3.5.3.1 del() [1/2]

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

```
                                        {
             auto it = cache_map.find(key);
              if (it == cache_map.end()) {
                  return false;  // Key not found
              }

              // Update memory usage
              current_memory_usage -= it->second->key.size() + it->second->value.size();

              // Remove from list and map
              cache_list.erase(it->second);
              cache_map.erase(it);

              return true;
          }
```

Here is the caller graph for this function:



3.5.3.2 del() [2/2]

```
int LRUCache::del (
              const void * key )  [inline]
```

Deletes a key-value pair from the cache.

Parameters

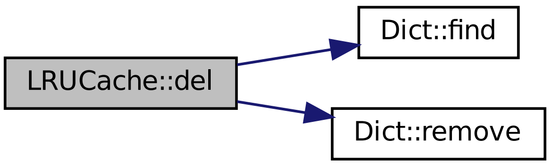
key	The key to delete

Returns

int 0 if successful, 1 if the key was not found

```
    {
        Node *retrievedValue = static_cast<Node *>(dict.find(key));

        if (retrievedValue)
        {
            remove(retrievedValue);
            current_memory_usage -= getSize((char *)retrievedValue->key) + getNodeSize(retrievedValue);
            dict.remove(retrievedValue->key);
            return 0;
        }
        return 1;
    }
```

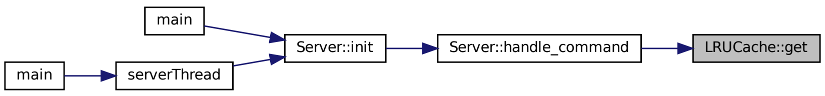
Here is the call graph for this function:



3.5.3.3 get() `[1/2]`

```
bool LRUCache::get (
               const std::string & key,
               std::string & value )   [inline]
                                                         {
            auto it = cache_map.find(key);
            if (it == cache_map.end()) {
                return false;  // Key not found
            }

            // Update access time and move to front of list
            it->second->last_accessed = std::chrono::steady_clock::now();

            // Move to front (most recently used)
            if (it->second != cache_list.begin()) {
                cache_list.splice(cache_list.begin(), cache_list, it->second);
            }

            value = it->second->value;
            return true;
        }
```

Here is the caller graph for this function:



3.5.3.4 get() [2/2]

```
std::string LRUCache::get (
              const void * key )  [inline]
```

Retrieves the value for a given key.

If the key exists, it moves the corresponding node to the front of the list to mark it as most recently used.

Parameters

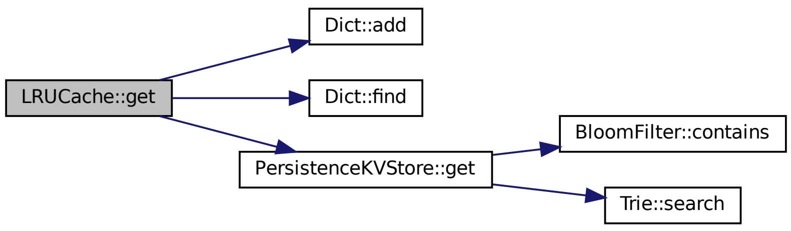
key	The key to look up

Returns

std::string The value associated with the key, or "-1" if not found

```
{
    Node *retrievedValue = static_cast<Node *>(dict.find(key));
    if (!retrievedValue){
        std::string value;
        if(storage.get(std::string((char *)key), value)){
            void *key1 = (void *)key;
            void* value1 = (void *) strdup(value.c_str());
            Node *node = new Node(key1, value1);
            dict.add(key1, node);
            add(node);
            current_memory_usage += getSize((char *)key) + getNodeSize(node);
             return value;
         };
         return "-1";
     }


     remove(retrievedValue);
     add(retrievedValue);
     return static_cast<char *>(retrievedValue->value);
 }
```

Here is the call graph for this function:



3.5.3.5 max_memory() [1/2]

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

Gets the maximum memory limit of the cache.

Returns

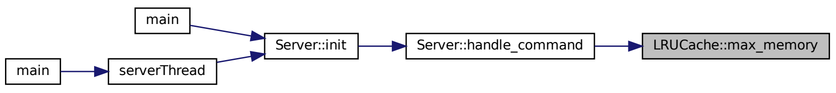
size_t Maximum memory limit in bytes

```
    {
        return max_memory_bytes;
    }
```

3.5.3.6 max_memory() [2/2]

```
size_t LRUCache::max_memory ( ) const  [inline]
                                      {
            return max_memory_bytes;
        }
```

Here is the caller graph for this function:



3.5.3.7 memory_usage() [1/2]

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

Gets the current memory usage of the cache.

Returns

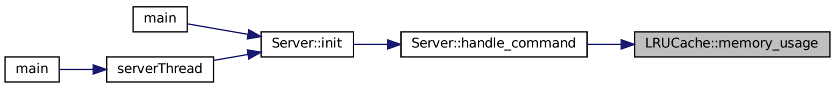
size_t Current memory usage in bytes

```
    {
        return current_memory_usage;
    }
```

3.5.3.8 memory_usage() [2/2]

```
size_t LRUCache::memory_usage ( ) const  [inline]
                                        {
            return current_memory_usage;
        }
```

Here is the caller graph for this function:



3.5.3.9 printList()

```
void LRUCache::printList ( )  [inline]
```

Prints the current state of the cache for debugging.

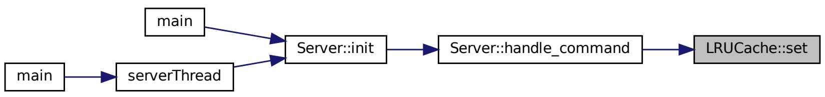
```
    {
        Node *curr = head->next;
        std::cout « "Cache state: ";
        while (curr != tail)
        {
            std::cout « "[" « static_cast<char *>(curr->key) « ":" « static_cast<char *>(curr->value) «
   "] ";
            curr = curr->next;
        }
        std::cout « std::endl;
    }
```

3.5.3.10 set() [1/2]

```
void LRUCache::set (
            const std::string & key,
            const std::string & value )  [inline]
```

```
                                                                        {
            // Calculate memory for this entry (key + value + overhead)
            size_t entry_size = key.size() + value.size() + sizeof(CacheEntry);

            // Check if key already exists
            auto it = cache_map.find(key);
            if (it != cache_map.end()) {
                // Update existing entry
                current_memory_usage -= it->second->key.size() + it->second->value.size();
                cache_list.erase(it->second);
                cache_map.erase(it);
            }

            // Check if we need to evict entries to make space
            while (current_memory_usage + entry_size > max_memory_bytes && !cache_list.empty()) {
                // Evict least recently used item
                auto last = cache_list.back();
                std::cout « "LRU Eviction: Removing key '" « last.key « "'" « std::endl;
                current_memory_usage -= last.key.size() + last.value.size();
                cache_map.erase(last.key);
                cache_list.pop_back();
            }

            // If we still can't fit the new entry, don't add it
            if (current_memory_usage + entry_size > max_memory_bytes) {
                std::cerr « "Warning: Entry too large to fit in cache" « std::endl;
                return;
            }

            // Add new entry to front of list (most recently used)
            cache_list.emplace_front(key, value);
            cache_map[key] = cache_list.begin();
            current_memory_usage += entry_size;

            // std::cout « "Memory usage: " « current_memory_usage « "/" « max_memory_bytes
            //           « " bytes (" « (current_memory_usage * 100.0 / max_memory_bytes) « "%)" «
     std::endl;
        }
```

Here is the caller graph for this function:



3.5.3.11 set() [2/2]

```
void LRUCache::set (
            void * key,
            void * value )  [inline]
```

Adds or updates a key-value pair in the cache.

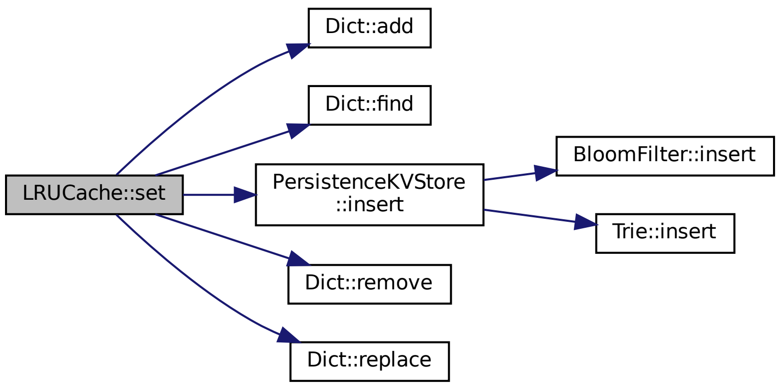
If adding the new item exceeds the memory capacity, the least recently used items will be evicted until the memory usage is within limits.

Parameters

key	Pointer to the key
value	Pointer to the value

```
{
    Node *retrievedValue = static_cast<Node *>(dict.find(key));
    Node *node = new Node(key, value);

    if (retrievedValue)
    {
        current_memory_usage += getNodeSize(node) - getNodeSize(retrievedValue);
        remove(retrievedValue);
        dict.replace(key, node);
        add(node);
    }
    else
    {
        dict.add(key, node);
        add(node);
        current_memory_usage += getSize((char *)key) + getNodeSize(node);
    }

    if (current_memory_usage >= max_memory_bytes)
    {
        Node *nodeToDelete = tail->prev;
        storage.insert(std::string(strdup((char *)nodeToDelete->key)), std::string(strdup((char
*)nodeToDelete->value)));
        remove(nodeToDelete);
        std::cout « current_memory_usage « std::endl;
        current_memory_usage -= getSize((char *)nodeToDelete->key) + getNodeSize(nodeToDelete);
        dict.remove(nodeToDelete->key);
        std::cout « "Eviction happen" « std::endl;
        std::cout « current_memory_usage « std::endl;
    }
}
```

Here is the call graph for this function:



3.5.3.12 size() [1/2]

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

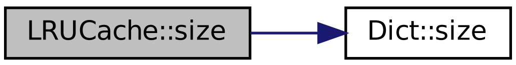
Gets the number of items in the cache.

Returns

size_t Number of items in the cache

```
    {
        return dict.size();
    }
```

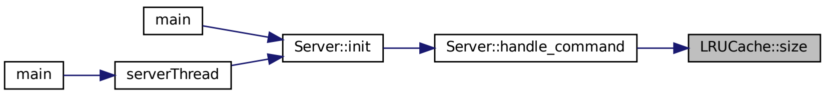
Here is the call graph for this function:



3.5.3.13 size() [2/2]

```
size_t LRUCache::size ( ) const  [inline]
                           {
            return cache_map.size();
        }
```

Here is the caller graph for this function:



3.5.4 Member Data Documentation

3.5.4.1 dict

`Dict LRUCache::dict`

Dictionary for O(1) lookups

3.5.4.2 head

`Node* LRUCache::head`

Head of the doubly linked list

3.5.4.3 storage

```
PersistenceKVStore LRUCache::storage
```

3.5.4.4 tail

```
Node* LRUCache::tail
```

Tail of the doubly linked list

3.5.4.5 value

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

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

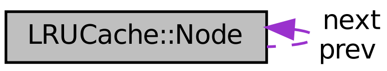
- lib/lru_cache_v0.h
- lib/lru_cache_v1.h

3.6 LRUCache::Node Struct Reference

Represents a node in the doubly linked list for the LRU cache.

```
#include <lru_cache_v1.h>
```

Collaboration diagram for LRUCache::Node:



Public Member Functions

- Node (void ∗k, void ∗v)

  Constructs a new Node with the given key and value.

Public Attributes

- void ∗ key
- void ∗ value
- Node ∗ next
- Node ∗ prev

3.6.1 Detailed Description

Represents a node in the doubly linked list for the LRU cache.

Each node contains a key-value pair and pointers to the next and previous nodes.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Node()

```
LRUCache::Node::Node (
             void * k,
             void * v )  [inline]
```

Constructs a new Node with the given key and value.

Parameters

k	Pointer to the key
v	Pointer to the value

```
        {
            key = k;
            value = v;
            next = nullptr;
            prev = nullptr;
        }
```

3.6.3 Member Data Documentation

3.6.3.1 key

```
void* LRUCache::Node::key
```

Pointer to the key

3.6.3.2 next

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

Pointer to the next node in the list

3.6.3.3 prev

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

Pointer to the previous node in the list

3.6.3.4 value

```
void* LRUCache::Node::value
```

Pointer to the value

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

- lib/lru_cache_v1.h

3.7 PersistenceKVStore Class Reference

A persistent key-value store with background rewriting and indexing.

```
#include <persistence_kv_store.h>
```

Public Member Functions

- PersistenceKVStore (const std::string _dbname, int _bloom_filter_size=10000, int rewrite_interval=5000)

  Constructor: Initializes the key-value store, loads the index, and starts the rewrite scheduler.
- ∼PersistenceKVStore ()

  Destructor: Ensures background rewriting stops and closes the file.
- void insert (const std::string &_key, const std::string &_value)

  Inserts a key-value pair into the store.
- bool get (const std::string &_key, std::string &_value)

  Retrieves the value for a given key.
- void remove (const std::string &_key)

  Removes a key from the store.
- void remove_db ()

  Removes the database file.

3.7.1 Detailed Description

A persistent key-value store with background rewriting and indexing.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 PersistenceKVStore()

```
PersistenceKVStore::PersistenceKVStore (
            const std::string _dbname,
            int _bloom_filter_size = 10000,
            int rewrite_interval = 5000 )
```

Constructor: Initializes the key-value store, loads the index, and starts the rewrite scheduler.

Parameters

_dbname	The database name.
_bloom_filter_size	The size of BloomFilter.
_rewrite_interval	Interval for background rewrite in milliseconds (default: 5000).

```
    : bloomFilter(_bloom_filter_size)
{
    filename = _dbname + ".txt";
    tempfilename = _dbname + ".temp.txt";
    rewrite_interval_ms = _rewrite_interval;
    stopRewrite.store(false);
    dataFile.open(filename, std::ios::in | std::ios::out | std::ios::binary);
     index = new Trie();

     if (!dataFile)
     {
         dataFile.open(filename, std::ios::out | std::ios::binary);
         dataFile.close();
         dataFile.open(filename, std::ios::in | std::ios::out | std::ios::binary);
     }

     syncIndex();
     rewriteThread = std::thread(&PersistenceKVStore::startRewriteScheduler, this);
 }
```

3.7.2.2 ∼PersistenceKVStore()

```
PersistenceKVStore::~PersistenceKVStore ( )
```

Destructor: Ensures background rewriting stops and closes the file.

```
 {
     dataFile.clear();
     stopRewrite.store(true);
     if (rewriteThread.joinable())
     {
         rewriteThread.join();
     }
     dataFile.close();
 }
```

3.7.3 Member Function Documentation

3.7.3.1 get()

```
bool PersistenceKVStore::get (
              const std::string & _key,
              std::string & _value )
```

Retrieves the value for a given key.

Parameters

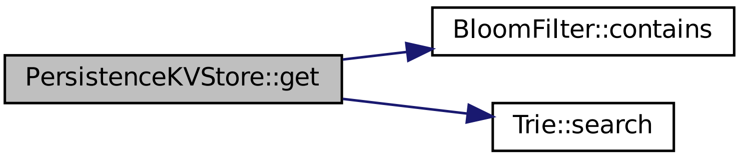
_key	The key to search for.
_value	Reference to store the retrieved value.

Returns

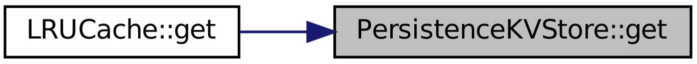
True if the key exists, false otherwise.

```
{
    std::lock_guard<std::mutex> lock(mtx_index);
    if (!bloomFilter.contains(_key))
        return false;

    long offset = index->search(_key);
    if (offset == -1)
        return false;

    dataFile.clear();
    dataFile.seekg(offset, std::ios::beg);

    if (!dataFile)
        return false;

    std::string storedKey;
    dataFile » storedKey;

    if (storedKey != _key)
        return false;

    std::getline(dataFile » std::ws, _value);

    return !_value.empty();
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.2 insert()

```
void PersistenceKVStore::insert (
            const std::string & _key,
            const std::string & _value )
```

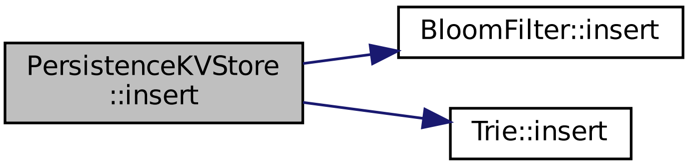
Inserts a key-value pair into the store.

Parameters

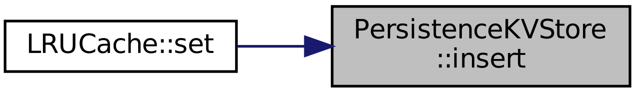
_key	The key to insert.
_value	The corresponding value.

```
{
    dataFile.clear();
    dataFile.seekp(0, std::ios::end);
    long offset = dataFile.tellp();
    if (offset == -1)
    {
        std::cerr « "Error: tellp() returned -1" « std::endl;
        return;
    }

    dataFile « _key « " " « _value « std::endl;
    dataFile.flush();
    std::lock_guard<std::mutex> lock(mtx_index);
    index->insert(_key, offset);
    bloomFilter.insert(_key);
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.3 remove()

```
void PersistenceKVStore::remove (
            const std::string & _key )
```

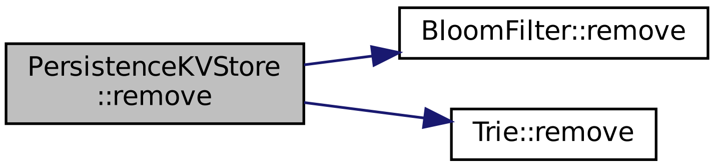
Removes a key from the store.

Parameters

_key	The key to remove.

```
{
    std::lock_guard<std::mutex> lock(mtx_index);
    index->remove(_key);
    bloomFilter.remove(_key);
}
```

Here is the call graph for this function:



3.7.3.4 remove_db()

```
void PersistenceKVStore::remove_db ( )
```

Removes the database file.

```
{
    std::remove(filename.c_str());
}
```

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

- lib/persistence_kv_store.h

3.8 Server Class Reference

Implements a Blink-compatible server with an LRU-based in-memory database.

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

Public Member Functions

- void parse_resp (const std::string &input, std::vector< std::string > &result)

  Parses a RESP (Redis Serialization Protocol) formatted string.
- void encode_resp (std::string &response, bool is_error)

  Encodes a response string into RESP format.
- void handle_command (const std::vector< std::string > &command, std::string &response)

  Handles client commands and generates appropriate responses.
- Server (std::string ip, int port, int buffer_size, int max_events, int max_mem_bytes)

  Constructs a Server object.
- ∼Server ()

  Destructor to release allocated resources.
- void init ()

  Initializes the server, sets up epoll, and starts listening for connections.

3.8.1 Detailed Description

Implements a Blink-compatible server with an LRU-based in-memory database.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Server()

```
Server::Server (
              std::string ip,
              int port,
              int buffer_size = 2048,
              int max_events = 4096,
              int max_mem_bytes = 1024 * 1024 * 1024 )
```

Constructs a Server object.

Parameters

ip	Server IP address.
port	Server port number.
buffer_size	Buffer size for receiving data.
max_events	Maximum epoll events.
max_mem_bytes	Maximum memory allocation for caching.

```
    : ip(_ip),
      database(_max_mem_bytes)
{
    port = _port;
    buffer_size = _buffer_size;
    max_mem_bytes = _max_mem_bytes;
    max_events = _max_events;
    tag = "[" + ip + ":" + std::to_string(_port) + "] ";
    buffer = new char[_buffer_size];
}
```

3.8.2.2 ∼Server()

```
Server::∼Server ( )
```

Destructor to release allocated resources.

```
{
    delete[] buffer;
}
```

3.8.3 Member Function Documentation

3.8.3.1 encode_resp()

```
void Server::encode_resp (
            std::string & response,
            bool is_error )
```

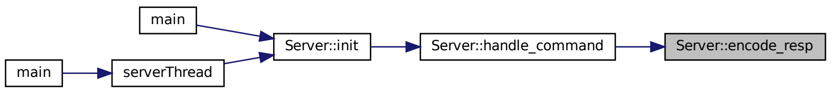
Encodes a response string into RESP format.

Parameters

response	Response string to encode.
is_error	Whether the response is an error message.

```
{
    if (is_error)
    {
        response = "-ERR " + response + "\r\n";
    }
    else if (response.empty())
    {
        response = "$-1\r\n";
    }
    else
    {
        response = "+" + response + "\r\n";
    }
}
```

Here is the caller graph for this function:



3.8.3.2 handle_command()

```
void Server::handle_command (
            const std::vector< std::string > & command,
            std::string & response )
```

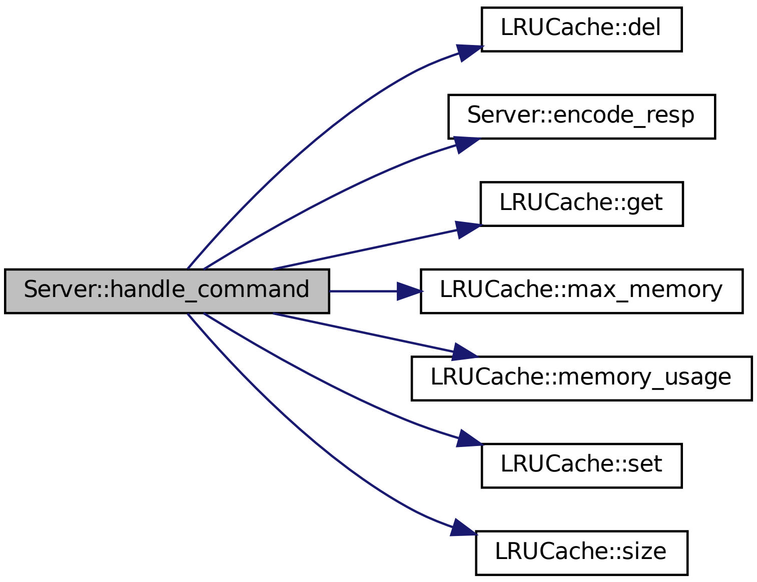
Handles client commands and generates appropriate responses.

Parameters

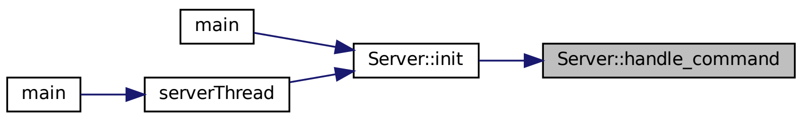
command	Parsed command tokens.
response	String to store the response.

```
{
    if (command.empty())
    {
        response = "Invalid command";
        encode_resp(response, true);
        return;
    }

    std::string cmd = command[0];
    std::transform(cmd.begin(), cmd.end(), cmd.begin(), ::toupper);

    if (cmd == "SET")
    {
        if (command.size() < 3)
        {
            response = "SET command requires key and value";
            encode_resp(response, true);
            return;
        }

        database.set(strdup(command[1].c_str()), strdup(command[2].c_str()));
        response = "OK";
        encode_resp(response, false);
    }
    else if (cmd == "GET")
    {
        if (command.size() < 2)
        {
            response = "GET command requires key";
            encode_resp(response, true);
            return;
        }

        std::string value = database.get(command[1].c_str());
        if (value != "-1")
        {
            response = "$" + std::to_string(value.length()) + "\r\n" + value + "\r\n";
        }
        else
        {
            response = "$-1\r\n";
        }
    }
    else if (cmd == "DEL")
    {
        if (command.size() < 2)
        {
            response = "DEL command requires key";
            encode_resp(response, true);
            return;
        }

        int count = 0;
        for (size_t i = 1; i < command.size(); i++)
        {
            count += database.del(command[i].c_str()) ? 1 : 0;
        }

        response = ":" + std::to_string(count) + "\r\n";
    }

    else if (cmd == "INFO")
    {
        // Add INFO command to get memory usage statistics
        std::string info = "# Memory\r\n";
        info += "used_memory:" + std::to_string(database.memory_usage()) + "\r\n";
        info += "maxmemory:" + std::to_string(database.max_memory()) + "\r\n";
        info += "maxmemory_policy:allkeys-lru\r\n";
        info += "# Stats\r\n";
        info += "keyspace_hits:" + std::to_string(database.size()) + "\r\n";

        response = "$" + std::to_string(info.length()) + "\r\n" + info + "\r\n";
    }
    else if (cmd == "CONFIG")
    {
        // Basic CONFIG command implementation
        if (command.size() < 2)
        {
            response = "CONFIG command requires subcommand";
            encode_resp(response, true);
            return;
        }

        std::string subcmd = command[1];
        std::transform(subcmd.begin(), subcmd.end(), subcmd.begin(), ::toupper);

        if (subcmd == "GET" && command.size() >= 3)
        {
            std::string param = command[2];
            std::transform(param.begin(), param.end(), param.begin(), ::tolower);

            if (param == "maxmemory")
            {
                response = "*2\r\n$9\r\nmaxmemory\r\n$" +
      std::to_string(std::to_string(database.max_memory()).length()) +
                           "\r\n" + std::to_string(database.max_memory()) + "\r\n";
                return;
            }
            else if (param == "maxmemory-policy")
            {
                response = "*2\r\n$16\r\nmaxmemory-policy\r\n$11\r\nallkeys-lru\r\n";
                return;
            }
        }
        response = "Supported CONFIG commands: GET maxmemory, GET maxmemory-policy";
        encode_resp(response, false);
    }

    else
    {
        response = "Unknown command";
        encode_resp(response, true);
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



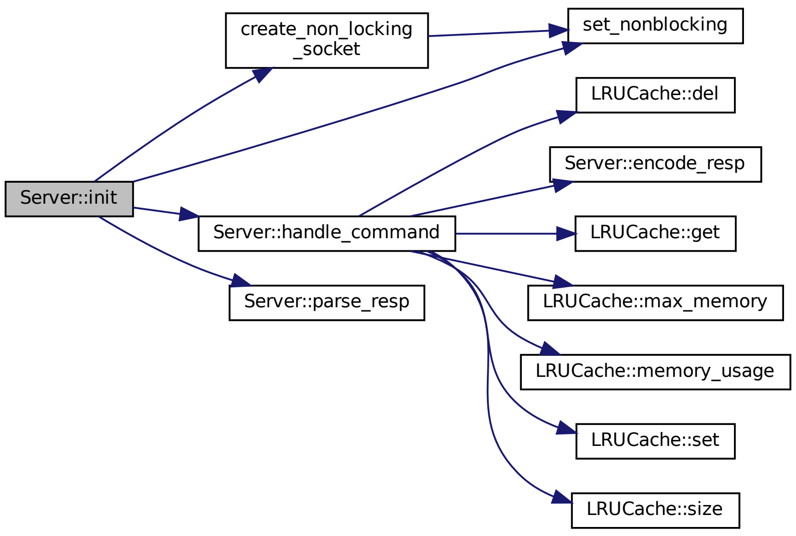
3.8.3.3 init()

```
void Server::init ( )
```

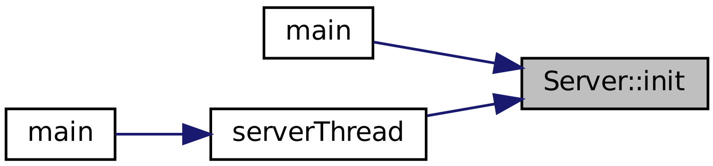
Initializes the server, sets up epoll, and starts listening for connections.

```
{
    int server_fd, epoll_fd;
    struct sockaddr_in address;
    socklen_t addrlen = sizeof(address);
    struct epoll_event event, events[max_events];

    server_fd = create_non_locking_socket(ip, port, address);

    epoll_fd = epoll_create1(0);
    if (epoll_fd == -1)
    {
        perror("[Server]: Epoll creation failed");
        exit(EXIT_FAILURE);
    }

    event.events = EPOLLIN;
    event.data.fd = server_fd;
    if (epoll_ctl(epoll_fd, EPOLL_CTL_ADD, server_fd, &event) == -1)
    {
        perror("[Server] Epoll_ctl failed");
        exit(EXIT_FAILURE);
    }

    std::cout « tag « "Blink-compatible server listening on port " « port « std::endl;
    std::cout « tag « "Memory limit set to " « (max_mem_bytes / (1024 * 1024)) « " MB with LRU eviction
policy" « std::endl;

    while (true)
    {
        int ready_fds = epoll_wait(epoll_fd, events, max_events, -1);
        if (ready_fds == -1)
        {
            perror("Epoll wait failed");
            break;
        }

        for (int i = 0; i < ready_fds; i++)
        {
            int sock_fd = events[i].data.fd;

            if (sock_fd == server_fd)
            {
                int client_fd = accept(server_fd, (struct sockaddr *)&address, &addrlen);

                // char client_ip[INET_ADDRSTRLEN];
                // inet_ntop(AF_INET, &address.sin_addr, client_ip, INET_ADDRSTRLEN);
                // int client_port = ntohs(address.sin_port);

                if (client_fd == -1)
                {
                    perror("Accept failed");
                    continue;
                }

                set_nonblocking(client_fd);

                event.events = EPOLLIN | EPOLLET;
                event.data.fd = client_fd;
                if (epoll_ctl(epoll_fd, EPOLL_CTL_ADD, client_fd, &event) == -1)
                {
                    perror("Epoll_ctl client add failed");
                    close(client_fd);
                    continue;
                }

                // std::cout « tag « "New client connected: " « client_ip « ":" « client_port «
       std::endl;
            }
            else
            {
                // memset(buffer, 0, buffer_size);
                int bytes_read = recv(sock_fd, buffer, buffer_size, 0);

                if (bytes_read > 0)
                {
                    buffer[bytes_read] = '\0';
                    std::string input(buffer, bytes_read);
                    std::vector<std::string> result;
                    std::string response;

                    parse_resp(input, result);
                    handle_command(result, response);

                    send(sock_fd, response.c_str(), response.length(), 0);
                }
                else
                {
                    // std::cout « tag « "Client " « sock_fd « " disconnected." « std::endl;
                    epoll_ctl(epoll_fd, EPOLL_CTL_DEL, sock_fd, nullptr);
                    close(sock_fd);
                }
            }
        }
    }

    close(server_fd);
    close(epoll_fd);
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.3.4 parse_resp()

```
void Server::parse_resp (
            const std::string & input,
            std::vector< std::string > & result )
```

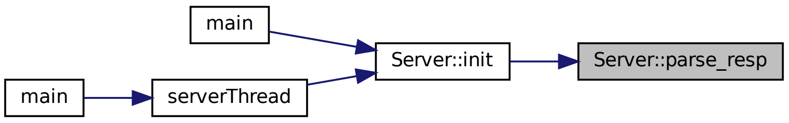
Parses a RESP (Redis Serialization Protocol) formatted string.

Parameters

input	Input string in RESP format.
result	Vector to store parsed tokens.

```
{
    if (input.empty())
        return;

    if (input[0] == '*')
    {
        size_t pos = 1;
        size_t newline = input.find("\r\n", pos);
        int array_len = std::stoi(input.substr(pos, newline - pos));

        pos = newline + 2;
        for (int i = 0; i < array_len; i++)
        {
            if (pos >= input.length())
                break;

            if (input[pos] == '$')
            {
                pos++;
                newline = input.find("\r\n", pos);
                int str_len = std::stoi(input.substr(pos, newline - pos));

                pos = newline + 2;
                result.push_back(input.substr(pos, str_len));
                pos += str_len + 2;
            }
        }
    }
}
```

Here is the caller graph for this function:



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

- lib/server.h

3.9 ServerAdd Struct Reference

```
#include <types.h>
```

Public Attributes

- int port
- std::string ip

3.9.1 Member Data Documentation

3.9.1.1 ip

```
std::string ServerAdd::ip
```

3.9.1.2 port

```
int ServerAdd::port
```

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

- lib/types.h

3.10 Trie Class Reference

```
#include <tire.h>
```

Public Member Functions

- Trie ()
- ∼Trie ()
- void insert (const std::string &key, long offset)
- long search (const std::string &key)
- void remove (const std::string &key)
- bool isDeleted (const std::string &key)

3.10.1 Constructor & Destructor Documentation

3.10.1.1 Trie()

```
Trie::Trie ( )  [inline]
22 { root = new TrieNode(); }
```

3.10.1.2 ∼Trie()

```
Trie::∼Trie ( )  [inline]
23 { deleteTrie(root); }
```

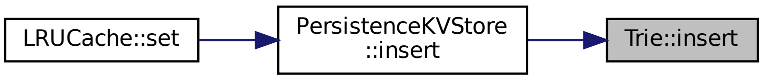
3.10.2 Member Function Documentation

3.10.2.1 insert()

```
void Trie::insert (
                const std::string & key,
                long offset )   [inline]
```

```
    {
        TrieNode *node = root;
        for (char ch : key)
        {
            if (!node->children.count(ch))
                node->children[ch] = new TrieNode();
            node = node->children[ch];
        }
        node->file_offset = offset;
        node->isDeleted = false;
    }
```

Here is the caller graph for this function:



3.10.2.2 isDeleted()

```
bool Trie::isDeleted (
                const std::string & key )   [inline]
```

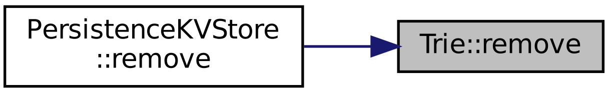
```
    {
        TrieNode *node = root;
        for (char ch : key)
        {
            if (!node->children.count(ch))
                return false;
            node = node->children[ch];
        }
        return node->isDeleted;
    }
```

3.10.2.3 remove()

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

```
    {
        TrieNode *node = root;
        for (char ch : key)
        {
            if (!node->children.count(ch))
                return;
            node = node->children[ch];
        }
        node->isDeleted = true;
    }
```

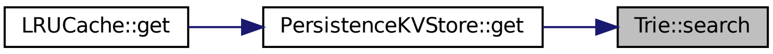
Here is the caller graph for this function:



3.10.2.4 search()

```
long Trie::search (
              const std::string & key )  [inline]
    {
        TrieNode *node = root;
        for (char ch : key)
        {
            if (!node->children.count(ch))
                return -1;
            node = node->children[ch];
        }
        return node->isDeleted ? -1 : node->file_offset;
    }
```

Here is the caller graph for this function:



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

- lib/tire.h

3.11 TrieNode Class Reference

```
#include <tire.h>
```

Public Attributes

- std::unordered_map< char, TrieNode ∗ > children
- long file_offset = -1
- bool isDeleted = false

3.11.1 Member Data Documentation

3.11.1.1 children

```
std::unordered_map<char, TrieNode *> TrieNode::children
```

3.11.1.2 file_offset

```
long TrieNode::file_offset = -1
```

3.11.1.3 isDeleted

```
bool TrieNode::isDeleted = false
```

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

- lib/tire.h

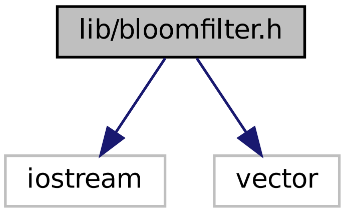
Chapter 4

File Documentation

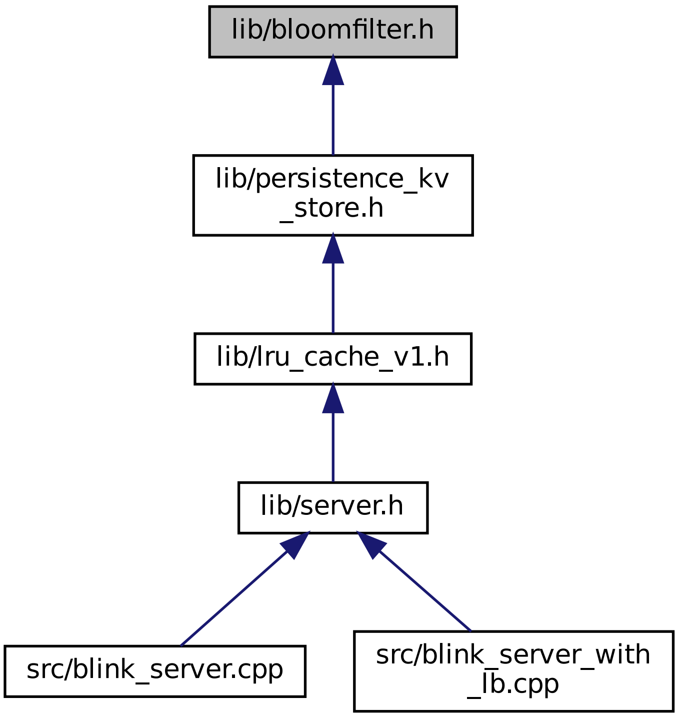
4.1 lib/bloomfilter.h File Reference

```
#include <iostream>
#include <vector>
```

Include dependency graph for bloomfilter.h:



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



Classes

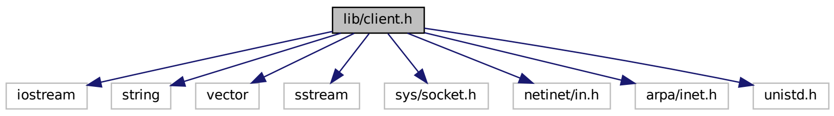
- class BloomFilter

  Implements a simple Bloom filter for fast key existence checks.

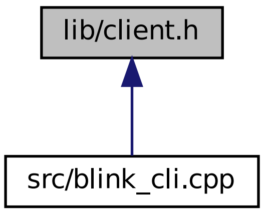
4.2 lib/client.h File Reference

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

Include dependency graph for client.h:



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



Classes

- class Client

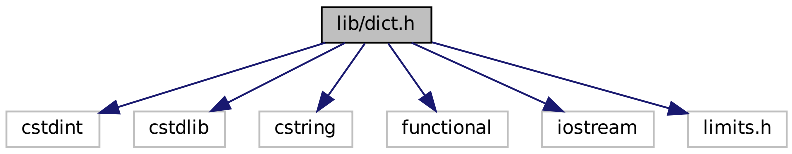
  A class to interact with a Blink server using RESP (Blink Serialization Protocol).

4.3 lib/dict.h File Reference

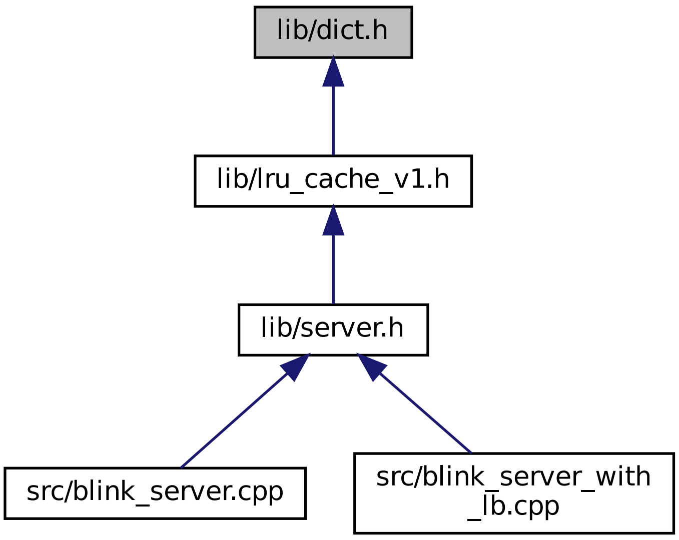
Implementation of a dictionary (hash table) with rehashing support.

```
#include <cstdint>
#include <cstdlib>
#include <cstring>
#include <functional>
#include <iostream>
#include <limits.h>
```

Include dependency graph for dict.h:



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



Classes

- class Dict

  A dictionary (hash table) implementation with dynamic resizing and rehashing.

Functions

- unsigned int stringHash (const void ∗key)

  Hash function for C-style strings.
- int stringCompare (const void ∗key1, const void ∗key2)

  Compares two C-style string keys.
- void ∗ stringDup (const void ∗key)

  Duplicates a C-style string key.
- void freeString (void ∗ptr)

  Frees a dynamically allocated C-style string.

4.3.1 Detailed Description

Implementation of a dictionary (hash table) with rehashing support.

4.3.2 Function Documentation

4.3.2.1 freeString()

```
void freeString (
            void * ptr )
```

Frees a dynamically allocated C-style string.

This function releases the memory allocated for a string key or value.

Parameters

ptr	Pointer to the string to be freed.

```
                        {
    free(ptr);
}
```

4.3.2.2 stringCompare()

```
int stringCompare (
            const void * key1,
            const void * key2 )
```

Compares two C-style string keys.

This function compares two string keys using `strcmp` and returns whether they are equal.

Parameters

key1	Pointer to the first string key.
key2	Pointer to the second string key.

Returns

int Returns 1 if keys are equal, 0 otherwise.

```
{
    return strcmp(static_cast<const char *>(key1), static_cast<const char *>(key2)) == 0;
}
```

4.3.2.3 stringDup()

```
void* stringDup (
            const void * key )
```

Duplicates a C-style string key.

This function creates a copy of the given string key using `strdup`. The caller is responsible for freeing the allocated memory.

Parameters

key	Pointer to the original string key.

Returns

void∗ Pointer to the duplicated string.

```
{
    return strdup(static_cast<const char *>(key));
}
```

4.3.2.4 stringHash()

```
unsigned int stringHash (
            const void * key )
```

Hash function for C-style strings.

This function computes a hash value for a given string using the DJB2 algorithm. It iterates through the characters and accumulates a hash value.

Parameters

key	Pointer to the C-style string key.

Returns

unsigned int The computed hash value.

```
{
    const char *str = static_cast<const char *>(key);
    unsigned int hash = 5381;
    int c;
    while ((c = *str++))
        hash = ((hash « 5) + hash) + c; // hash * 33 + c
    return hash;
}
```

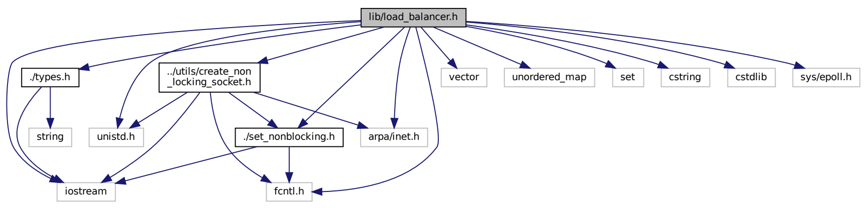
4.4 lib/load_balancer.h File Reference

Implements a non-blocking load balancer using epoll and consistent hashing.

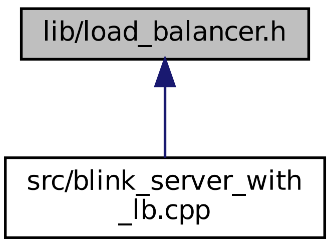
```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <set>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/epoll.h>
```

```
#include "../utils/create_non_locking_socket.h"
#include "../utils/set_nonblocking.h"
#include "./types.h"
```

Include dependency graph for load_balancer.h:



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



Classes

- class LoadBalancer

  Handles load balancing by distributing client requests to backend servers.

4.4.1 Detailed Description

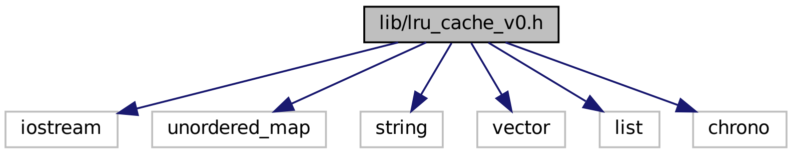
Implements a non-blocking load balancer using epoll and consistent hashing.

4.5 lib/lru_cache_v0.h File Reference

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

```
#include <chrono>
```

Include dependency graph for lru_cache_v0.h:



Classes

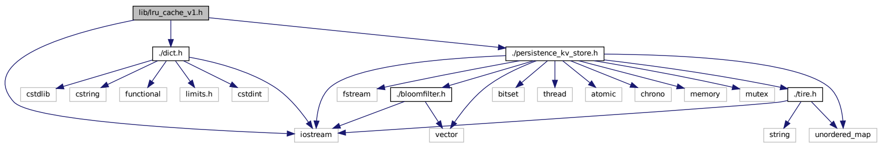
- class LRUCache

  Implements a Least Recently Used (LRU) cache with memory constraints.

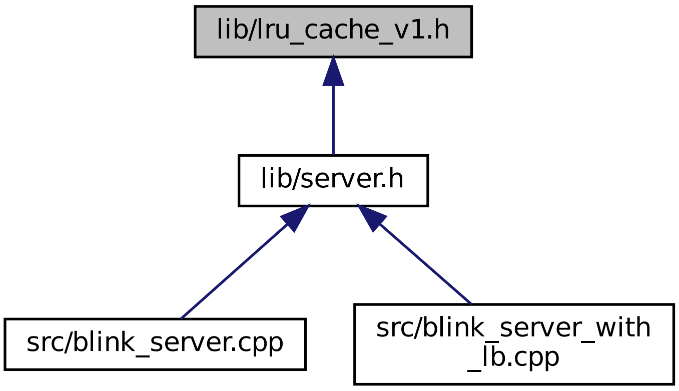
4.6 lib/lru_cache_v1.h File Reference

```
#include <iostream>
#include "./dict.h"
#include "./persistence_kv_store.h"
```

Include dependency graph for lru_cache_v1.h:



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



Classes

- class LRUCache

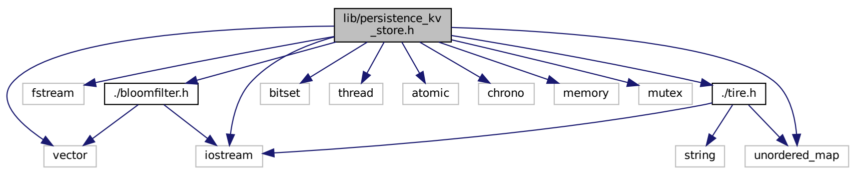
  Implements a Least Recently Used (LRU) cache with memory constraints.
- struct LRUCache::Node

  Represents a node in the doubly linked list for the LRU cache.

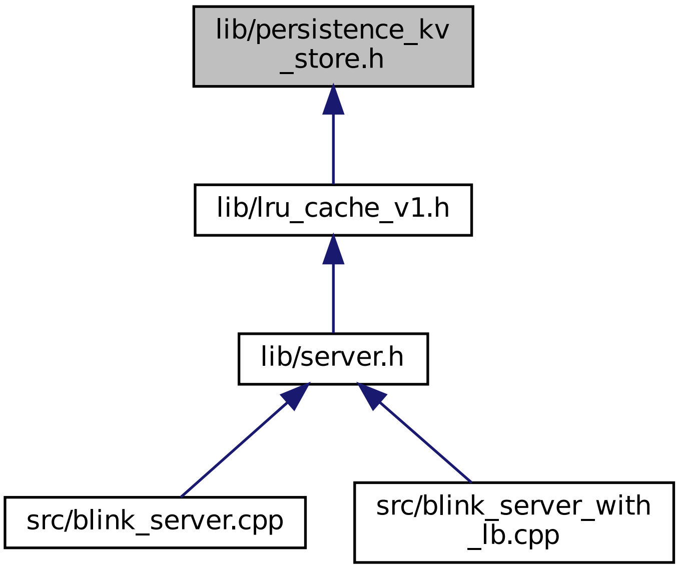
4.7 lib/persistence_kv_store.h File Reference

```
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <vector>
#include <bitset>
#include <thread>
#include <atomic>
#include <chrono>
#include <memory>
#include <mutex>
#include "./bloomfilter.h"
#include "./tire.h"
```

Include dependency graph for persistence_kv_store.h:



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



Classes

- class PersistenceKVStore

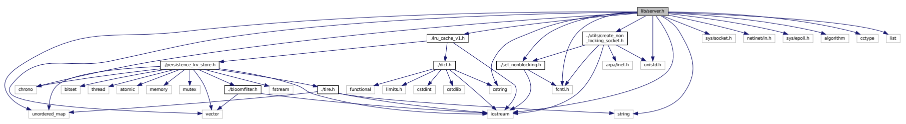
  A persistent key-value store with background rewriting and indexing.

4.8 lib/server.h File Reference

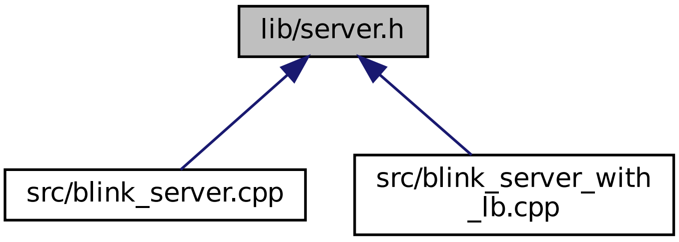
```
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/epoll.h>
#include <fcntl.h>
#include <unordered_map>
#include <string>
#include <vector>
#include <algorithm>
#include <cctype>
#include <list>
#include <chrono>
#include "./lru_cache_v1.h"
#include "../utils/create_non_locking_socket.h"
```

```
#include "../utils/set_nonblocking.h"
```

Include dependency graph for server.h:



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



Classes

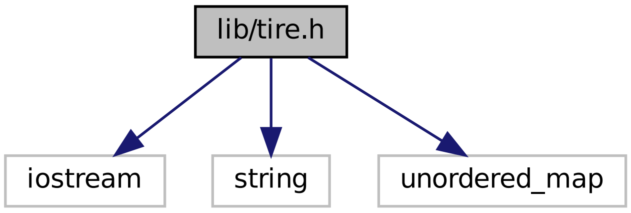
- class Server

  Implements a Blink-compatible server with an LRU-based in-memory database.

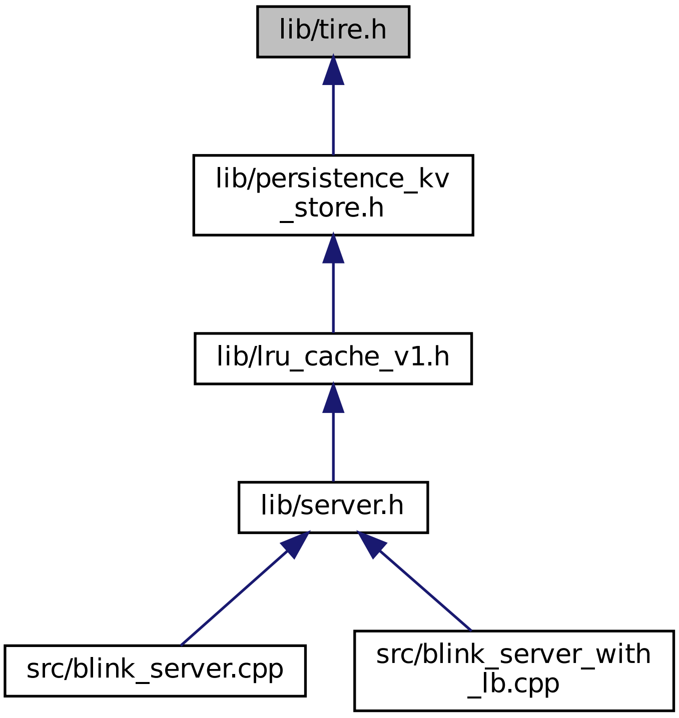
4.9 lib/tire.h File Reference

```
#include <iostream>
#include <string>
#include <unordered_map>
```

Include dependency graph for tire.h:



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



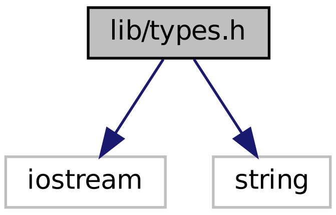
Classes

- class TrieNode
- class Trie

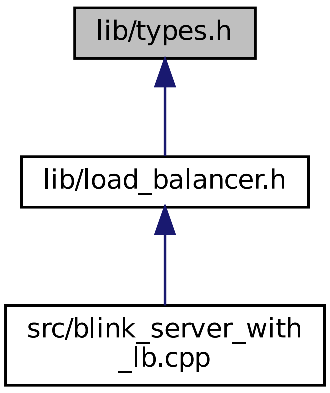
4.10 lib/types.h File Reference

```
#include <iostream>
#include <string>
```

Include dependency graph for types.h:



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



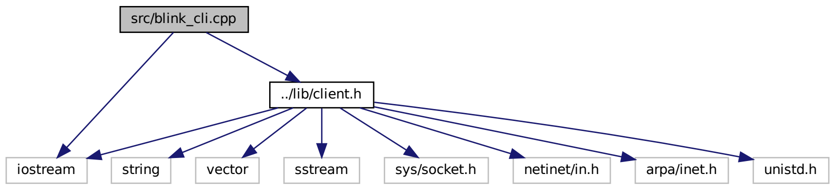
Classes

- struct ServerAdd

4.11 src/blink_cli.cpp File Reference

```
#include <iostream>
#include "../lib/client.h"
```

Include dependency graph for blink_cli.cpp:



Macros

- #define BUFFER_SIZE 1024
- #define SERVER_PORT 9001
- #define SERVER_IP "127.0.0.1"

Functions

- void command_loop (Client &client)

    Reads user input, parses commands, and interacts with the Client class.
- int main ()

4.11.1 Macro Definition Documentation

4.11.1.1 BUFFER_SIZE

```
#define BUFFER_SIZE 1024
```

4.11.1.2 SERVER_IP

```
#define SERVER_IP "127.0.0.1"
```

4.11.1.3 SERVER_PORT

```
#define SERVER_PORT 9001
```

4.11.2 Function Documentation

4.11.2.1 command_loop()

```
void command_loop (
             Client & client )
```

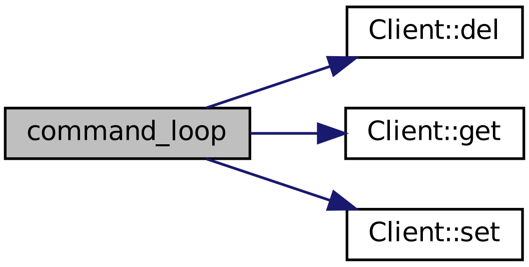
Reads user input, parses commands, and interacts with the Client class.

Parameters

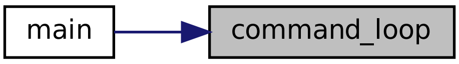
client	A reference to the Client object.

```
{
    std::cout « "Server is connected at " « client.ip_addr « ":" « client.port « std::endl;
    std::cout « "Enter command (SET key value, GET key, DEL key, or EXIT to quit):" « std::endl;

    std::string input, command, key, value;
    while (true)
    {
        std::cout « "> ";
        std::getline(std::cin, input);

        std::istringstream iss(input);
        iss » command;
        if (command == "SET")
        {
            iss » key » value;
            if (key.empty() || value.empty())
            {
                std::cout « "Invalid SET command. Usage: SET <key> <value>" « std::endl;
                continue;
            }
            std::cout « client.set(key, value) « std::endl;
        }
        else if (command == "GET")
        {
            iss » key;
            if (key.empty())
            {
                std::cout « "Invalid GET command. Usage: GET <key>" « std::endl;
                continue;
            }
            std::cout « client.get(key) « std::endl;
        }
        else if (command == "DEL")
        {
            iss » key;
            if (key.empty())
            {
                std::cout « "Invalid DEL command. Usage: DEL <key>" « std::endl;
                continue;
            }
            std::cout « client.del(key) « std::endl;
        }
        else if (command == "EXIT")
        {
            break;
        }
        else
        {
            std::cout « "Unknown command. Use SET, GET, DEL, or EXIT." « std::endl;
        }
    }
}
```

Here is the call graph for this function:



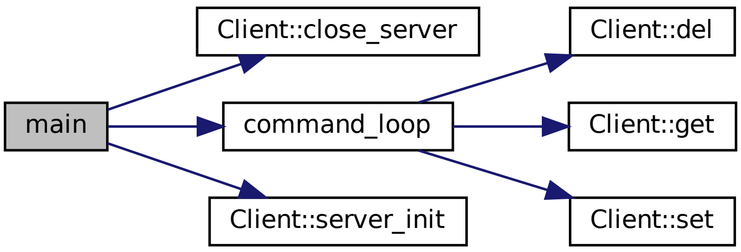
Here is the caller graph for this function:



4.11.2.2 main()

```
int main ( )
{
    Client client(SERVER_IP, SERVER_PORT, BUFFER_SIZE);

    if (client.server_init() == -1)
    {
        return -1;
    }

    command_loop(client);

    client.close_server();

    return 0;
}
```

Here is the call graph for this function:

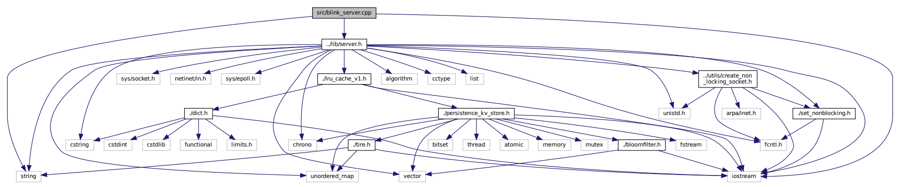


4.12 src/blink_server.cpp File Reference

```
#include <iostream>
#include <string>
```

```
#include "../lib/server.h"
```

Include dependency graph for blink_server.cpp:



Functions

- int main ()

  Main function to start the server.

Variables

- const int MAX_EVENTS = 4096

  Maximum number of events for epoll.
- const int BUFFER_SIZE = 2048

  Buffer size for reading data.
- const int SERVER_PORT = 9001

  Server port to bind to.
- const int MAX_MEMORY_BYTES = 1024 ∗ 1024 ∗ 1024

  Maximum memory allocation in bytes.
- const std::string SERVER_IP = "127.0.0.1"

  Server IP address.

4.12.1 Function Documentation

4.12.1.1 main()

```
int main ( )
```

Main function to start the server.

The server initializes and starts listening for incoming client connections.

Returns

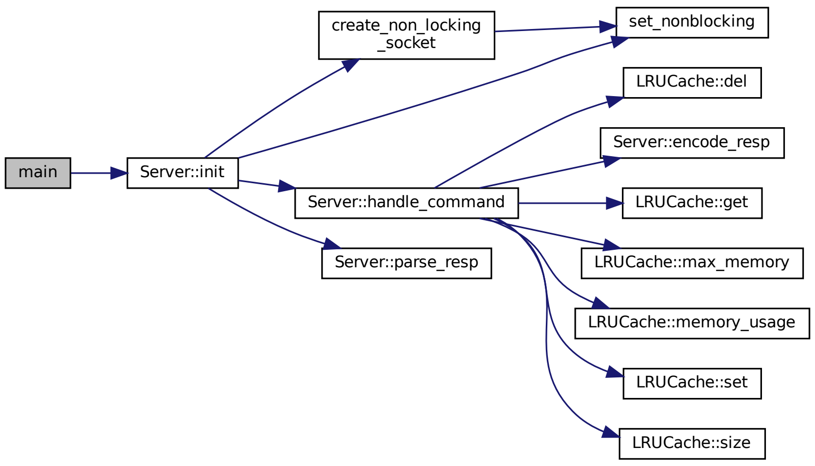
int Returns 0 on successful execution.

Create a Server instance with configuration parameters

Initialize the server

```
        {
    Server server(SERVER_IP, SERVER_PORT, BUFFER_SIZE, MAX_EVENTS, MAX_MEMORY_BYTES);

    server.init();

    return 0;
}
```

Here is the call graph for this function:



4.12.2 Variable Documentation

4.12.2.1 BUFFER_SIZE

```
const int BUFFER_SIZE = 2048
```

Buffer size for reading data.

4.12.2.2 MAX_EVENTS

```
const int MAX_EVENTS = 4096
```

Maximum number of events for epoll.

4.12.2.3 MAX_MEMORY_BYTES

```
const int MAX_MEMORY_BYTES = 1024 * 1024 * 1024
```

Maximum memory allocation in bytes.

4.12.2.4 SERVER_IP

```
const std::string SERVER_IP = "127.0.0.1"
```

Server IP address.

4.12.2.5 SERVER_PORT

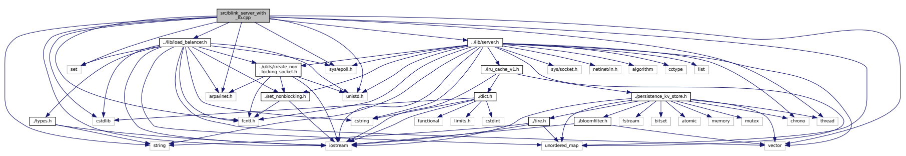
```
const int SERVER_PORT = 9001
```

Server port to bind to.

4.13 src/blink_server_with_lb.cpp File Reference

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <set>
#include <string>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/epoll.h>
#include <thread>
#include "../lib/server.h"
#include "../lib/load_balancer.h"
```

Include dependency graph for blink_server_with_lb.cpp:



Macros

- #define MAX_EVENTS 100
- #define BUFFER_SIZE 1024
- #define SERVER_PORT 9001
- #define MAX_MEMORY_BYTES 1024 ∗ 1024 ∗ 10
- #define SERVER_IP "127.0.0.1"

Functions

- void serverThread (std::string ip, int port)
- std::string parse_key (char ∗buffer, int bytes_read)
- int main ()

4.13.1 Macro Definition Documentation

4.13.1.1 BUFFER_SIZE

```
#define BUFFER_SIZE 1024
```

4.13.1.2 MAX_EVENTS

```
#define MAX_EVENTS 100
```

4.13.1.3 MAX_MEMORY_BYTES

```
#define MAX_MEMORY_BYTES 1024 ∗ 1024 ∗ 10
```

4.13.1.4 SERVER_IP

```
#define SERVER_IP "127.0.0.1"
```

4.13.1.5 SERVER_PORT

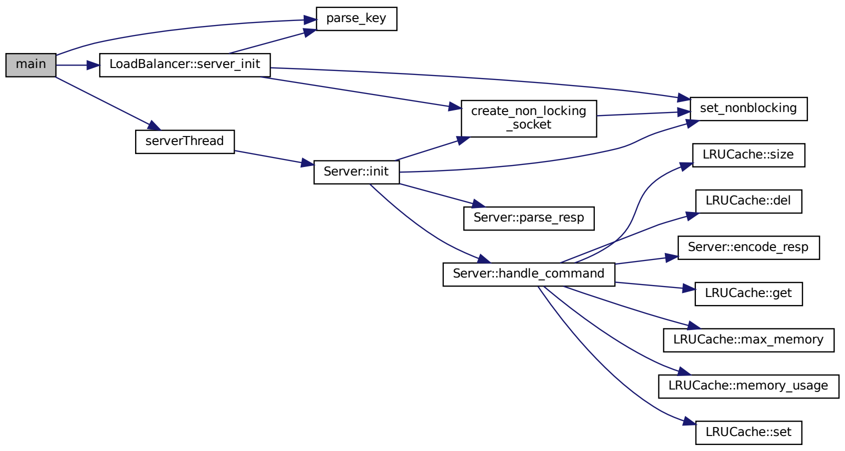
```
#define SERVER_PORT 9001
```

4.13.2 Function Documentation

4.13.2.1 main()

```
int main ( )
{

    int num_servers;
    std::cout « "Enter number of servers: ";
    std::cin » num_servers;

    std::vector<ServerAdd> servers_addr;
    for (int i = 0; i < num_servers; ++i) {
        ServerAdd server_add;
        server_add.port = 5000 + i;
        server_add.ip = SERVER_IP;
        servers_addr.push_back(server_add);
    }

    std::vector<std::thread> server_threads;
    for (auto &server_addr : servers_addr) {
        server_threads.emplace_back(serverThread, server_addr.ip, server_addr.port);
    }

    LoadBalancer loadBalancer(SERVER_IP, SERVER_PORT, servers_addr, BUFFER_SIZE, MAX_EVENTS);

    loadBalancer.server_init(&parse_key);

    for (auto& t : server_threads) t.join();

    return 0;
}
```

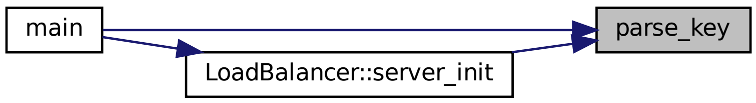
Here is the call graph for this function:



4.13.2.2 parse_key()

```
std::string parse_key (
            char * buffer,
            int bytes_read )
{

    std::string input(buffer, bytes_read);
    std::vector<std::string> result;

    if (input.empty())
        return "";

    if (input[0] == '*')
    {
        size_t pos = 1;
        size_t newline = input.find("\r\n", pos);
        int array_len = std::stoi(input.substr(pos, newline - pos));

        pos = newline + 2;
        for (int i = 0; i < array_len; i++)
        {
            if (pos >= input.length())
                break;

            if (input[pos] == '$')
            {
                pos++;
                newline = input.find("\r\n", pos);
                int str_len = std::stoi(input.substr(pos, newline - pos));

                pos = newline + 2;
                result.push_back(input.substr(pos, str_len));
                pos += str_len + 2;
            }
        }
    }

    return result[1];
}
```

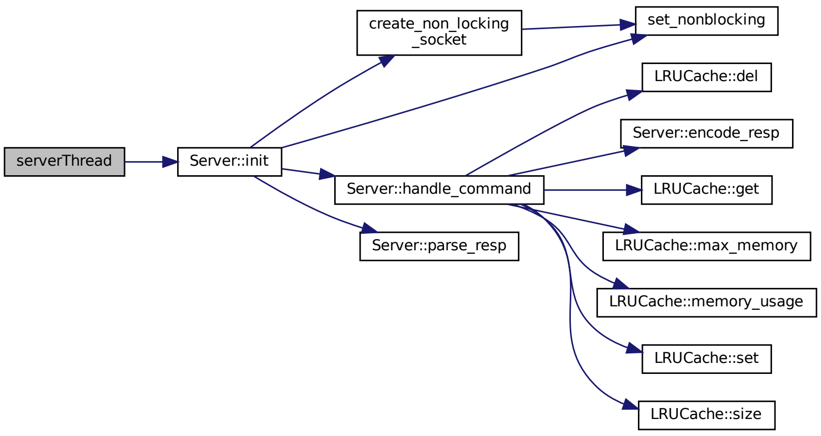
Here is the caller graph for this function:



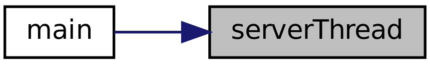
4.13.2.3 serverThread()

```
void serverThread (
              std::string ip,
              int port )
                                    {
    Server server(ip, port, BUFFER_SIZE, MAX_EVENTS, MAX_MEMORY_BYTES);
    server.init();
}
```

Here is the call graph for this function:



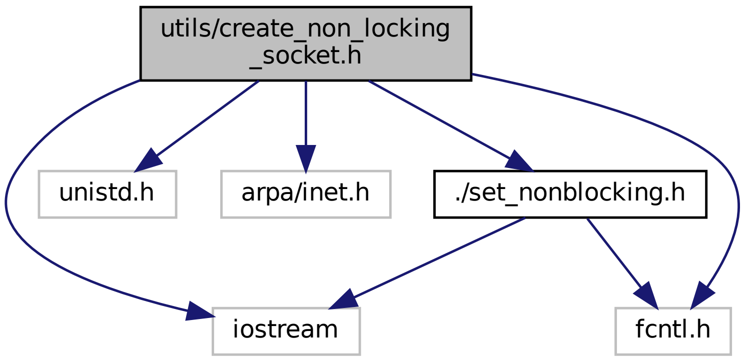
Here is the caller graph for this function:



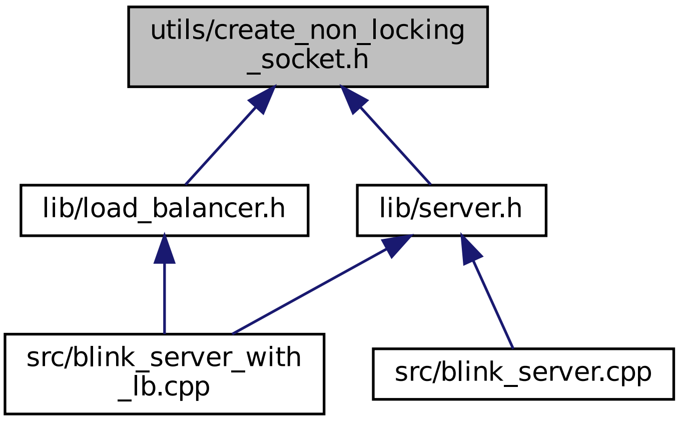
4.14 utils/create_non_locking_socket.h File Reference

```
#include <iostream>
#include <unistd.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include "./set_nonblocking.h"
```

Include dependency graph for create_non_locking_socket.h:



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



Functions

- int create_non_locking_socket (const std::string ip, const int port, struct sockaddr_in &addr)

  Creates a non-blocking socket.

4.14.1 Function Documentation

4.14.1.1 create_non_locking_socket()

```
int create_non_locking_socket (
              const std::string ip,
              const int port,
              struct sockaddr_in & addr )
```

Creates a non-blocking socket.

Parameters

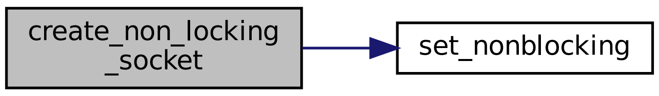
ip	The ip to bind the socket to.
port	The port number to bind the socket to.
addr	The addr to save the socket address.

Returns

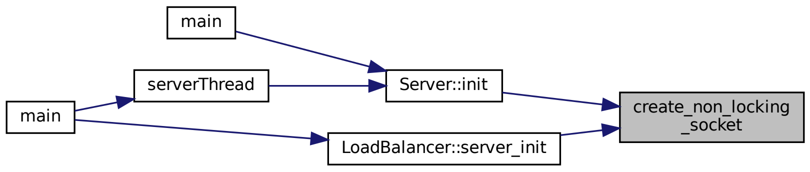
The socket file descriptor.

```
{
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    int opt = 1;
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

    if (sockfd < 0)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    set_nonblocking(sockfd);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);

    if (inet_pton(AF_INET, ip.c_str(), &addr.sin_addr) <= 0) {
        perror("Invalid IP address");
        exit(EXIT_FAILURE);
    }

    if (bind(sockfd, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(sockfd, SOMAXCONN) < 0)
    {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    return sockfd;
}
```

Here is the call graph for this function:



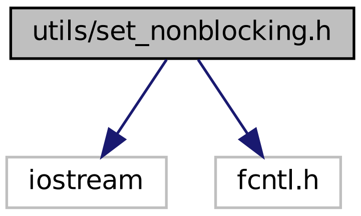
Here is the caller graph for this function:



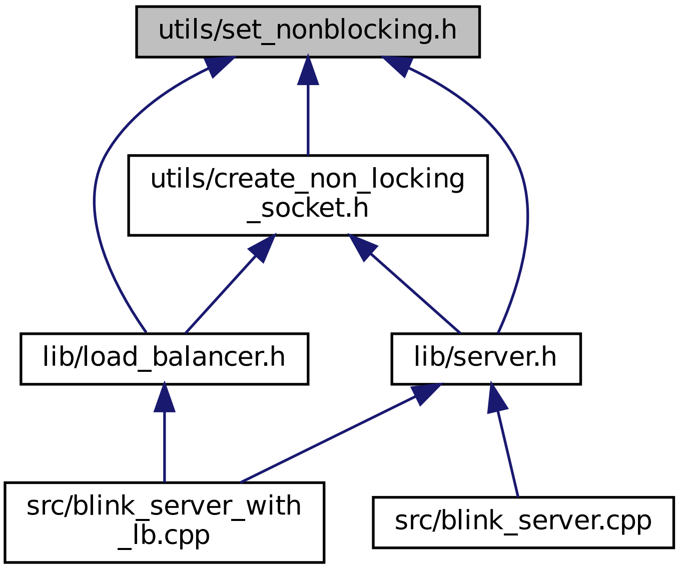
4.15 utils/set_nonblocking.h File Reference

```
#include <iostream>
#include <fcntl.h>
```

Include dependency graph for set_nonblocking.h:



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



Functions

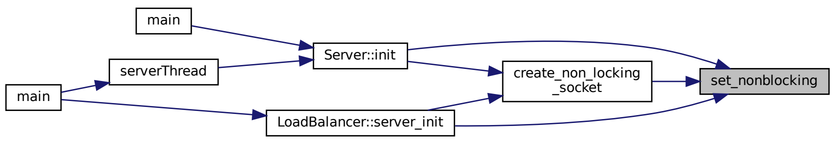
- void set_nonblocking (int sock)

4.15.1 Function Documentation

4.15.1.1 set_nonblocking()

```
void set_nonblocking (
            int sock )
{
    int flags = fcntl(sock, F_GETFL, 0);
    if (flags == -1)
    {
        perror("fcntl F_GETFL failed");
        exit(EXIT_FAILURE);
    }

    if (fcntl(sock, F_SETFL, flags | O_NONBLOCK) == -1)
    {
        perror("fcntl F_SETFL failed");
        exit(EXIT_FAILURE);
    }
}
```

Here is the caller graph for this function:



Index