## Saurion

Generated by Doxygen 1.9.4

| 1 Todo List                          | 1  |
|--------------------------------------|----|
| 2 Module Index                       | 3  |
| 2.1 Modules                          | 3  |
| 3 Class Index                        | 5  |
| 3.1 Class List                       | 5  |
| 4 File Index                         | 7  |
| 4.1 File List                        | 7  |
| 7.11 NO EIGHT                        | ,  |
| 5 Module Documentation               | 9  |
| 5.1 LowSaurion                       | 9  |
| 5.1.1 Detailed Description           | 10 |
| 5.1.2 Macro Definition Documentation | 12 |
| 5.1.2.1 _POSIX_C_SOURCE              | 12 |
| 5.1.2.2 PACKING_SZ                   | 12 |
| 5.1.3 Function Documentation         | 12 |
| 5.1.3.1 allocate_iovec()             | 13 |
| 5.1.3.2 EXTERNAL_set_socket()        | 13 |
| 5.1.3.3 free_request()               | 14 |
| 5.1.3.4 initialize_iovec()           | 14 |
| 5.1.3.5 read_chunk()                 | 15 |
| 5.1.3.6 saurion_create()             | 19 |
| 5.1.3.7 saurion_destroy()            | 21 |
| 5.1.3.8 saurion_send()               | 22 |
| 5.1.3.9 saurion_start()              | 22 |
| 5.1.3.10 saurion_stop()              | 23 |
| 5.1.3.11 set_request()               | 23 |
| 5.2 ThreadPool                       | 25 |
| 5.2.1 Detailed Description           | 25 |
| 5.2.2 Function Documentation         | 25 |
| 5.2.2.1 threadpool_add()             | 26 |
| 5.2.2.2 threadpool_create()          | 26 |
| 5.2.2.3 threadpool_create_default()  | 27 |
| 5.2.2.4 threadpool_destroy()         | 27 |
| . –                                  |    |
| 5.2.2.5 threadpool_empty()           | 28 |
| 5.2.2.6 threadpool_init()            | 28 |
| 5.2.2.7 threadpool_stop()            | 29 |
| 5.2.2.8 threadpool_wait_empty()      | 29 |
| 6 Class Documentation                | 31 |
| 6.1 Node Struct Reference            | 31 |
| 6.1.1 Detailed Description           | 31 |

| 6.1.2 Member Data Documentation              | 31 |
|--|----|
| 6.1.2.1 children                             | 31 |
| 6.1.2.2 next                                 | 31 |
| 6.1.2.3 ptr                                  | 32 |
| 6.1.2.4 size                                 | 32 |
| 6.2 request Struct Reference                 | 32 |
| 6.2.1 Detailed Description                   | 32 |
| 6.2.2 Member Data Documentation              | 32 |
| 6.2.2.1 client_socket                        | 32 |
| 6.2.2.2 event_type                           | 33 |
| 6.2.2.3 iov                                  | 33 |
| 6.2.2.4 iovec_count                          | 33 |
| 6.2.2.5 next_iov                             | 33 |
| 6.2.2.6 next_offset                          | 33 |
| 6.2.2.7 prev                                 | 33 |
| 6.2.2.8 prev_remain                          | 34 |
| 6.2.2.9 prev_size                            | 34 |
| 6.3 saurion Struct Reference                 | 34 |
| 6.3.1 Detailed Description                   | 35 |
| 6.3.2 Member Data Documentation              | 35 |
| 6.3.2.1 efds                                 | 35 |
| 6.3.2.2 list                                 | 35 |
| 6.3.2.3 m_rings                              | 35 |
| 6.3.2.4 n_threads                            | 35 |
| 6.3.2.5 next                                 | 36 |
| 6.3.2.6 pool                                 | 36 |
| 6.3.2.7 rings                                | 36 |
| 6.3.2.8 ss                                   | 36 |
| 6.3.2.9 status                               | 36 |
| 6.3.2.10 status_c                            | 37 |
| 6.3.2.11 status_m                            | 37 |
| 6.4 Saurion Class Reference                  | 37 |
| 6.4.1 Detailed Description                   | 38 |
| 6.4.2 Member Typedef Documentation           | 38 |
| 6.4.2.1 ClosedCb                             | 38 |
| 6.4.2.2 ConnectedCb                          | 38 |
| 6.4.2.3 ErrorCb                              | 38 |
| 6.4.2.4 ReadedCb                             | 38 |
| 6.4.2.5 WroteCb                              | 39 |
| 6.4.3 Constructor & Destructor Documentation | 39 |
| <b>6.4.3.1 Saurion()</b> [1/3]               | 39 |
| 6.4.3.2 ~Saurion()                           | 39 |

| <b>6.4.3.3 Saurion()</b> [2/3]                  | 39 |
|---|----|
| <b>6.4.3.4 Saurion()</b> [3/3]                  | 39 |
| 6.4.4 Member Function Documentation             | 40 |
| 6.4.4.1 init()                                  | 40 |
| 6.4.4.2 on_closed()                             | 40 |
| 6.4.4.3 on_connected()                          | 40 |
| 6.4.4.4 on_error()                              | 40 |
| 6.4.4.5 on_readed()                             | 41 |
| 6.4.4.6 on_wrote()                              | 41 |
| 6.4.4.7 operator=() [1/2]                       | 41 |
| 6.4.4.8 operator=() [2/2]                       | 41 |
| 6.4.4.9 send()                                  | 41 |
| 6.4.4.10 stop()                                 | 42 |
| 6.4.5 Member Data Documentation                 | 42 |
| 6.4.5.1 s                                       | 42 |
| 6.5 saurion::saurion_callbacks Struct Reference | 42 |
| 6.5.1 Detailed Description                      | 43 |
| 6.5.2 Member Data Documentation                 | 43 |
| 6.5.2.1 on_closed                               | 43 |
| 6.5.2.2 on_closed_arg                           | 43 |
| 6.5.2.3 on_connected                            | 43 |
| 6.5.2.4 on_connected_arg                        | 44 |
| 6.5.2.5 on_error                                | 44 |
| 6.5.2.6 on_error_arg                            | 44 |
| 6.5.2.7 on_readed                               | 44 |
| 6.5.2.8 on_readed_arg                           | 45 |
| 6.5.2.9 on_wrote                                | 45 |
| 6.5.2.10 on_wrote_arg                           | 45 |
| 6.6 saurion_callbacks Struct Reference          | 46 |
| 6.6.1 Detailed Description                      | 46 |
| 6.6.2 Member Data Documentation                 | 46 |
| 6.6.2.1 on_closed                               | 46 |
| 6.6.2.2 on_closed_arg                           | 47 |
| 6.6.2.3 on_connected                            | 47 |
| 6.6.2.4 on_connected_arg                        | 47 |
| 6.6.2.5 on_error                                | 47 |
| 6.6.2.6 on_error_arg                            | 48 |
| 6.6.2.7 on_readed                               | 48 |
| 6.6.2.8 on_readed_arg                           | 48 |
| 6.6.2.9 on_wrote                                | 48 |
| 6.6.2.10 on_wrote_arg                           | 49 |
| 6.7 saurion, wrapper Struct Reference           | 49 |

|   | 6.7.1 Detailed Description                                   | 49                   |
|---|--|----------------------|
|   | 6.7.2 Member Data Documentation                              | 49                   |
|   | 6.7.2.1 s  | 49                   |
|   | 6.7.2.2 sel  | 50                   |
|   | 6.8 task Struct Reference                                    | 50                   |
|   | 6.8.1 Detailed Description                                   | 50                   |
|   | 6.8.2 Member Data Documentation                              | 50                   |
|   | 6.8.2.1 argument   | 50                   |
|   | 6.8.2.2 function   | 50                   |
|   | 6.8.2.3 next   | 51                   |
|   | 6.9 threadpool Struct Reference                              | 51                   |
|   | 6.9.1 Detailed Description                                   | 51                   |
|   | 6.9.2 Member Data Documentation                              | 51                   |
|   | 6.9.2.1 empty_cond   | 51                   |
|   | 6.9.2.2 num_threads  | 52                   |
|   | 6.9.2.3 queue_cond   | 52                   |
|   | 6.9.2.4 queue_lock   | 52                   |
|   | 6.9.2.5 started  | 52                   |
|   | 6.9.2.6 stop   | 52                   |
|   | 6.9.2.7 task_queue_head                                      | 52                   |
|   | 6.9.2.8 task_queue_tail                                      | 52                   |
|   | 6.9.2.9 threads  | 52                   |
| 7 | File Documentation   | 53                   |
| • | 7.1 /_w/saurion/saurion/include/linked_list.h File Reference | 53                   |
|   | 7.1.1 Function Documentation                                 | 53                   |
|   | 7.1.1.1 list delete node()                                   | 53                   |
|   | 7.1.1.2 list free()  |                      |
|   | 7.1.1.3 list_insert()  | 54                   |
|   | 7.2 linked_list.h  | 55                   |
|   | 7.3 /w/saurion/saurion/include/low_saurion.h File Reference  | 55                   |
|   | 7.3.1 Variable Documentation                                 | 56                   |
|   | 7.3.1.1 efds   | 56                   |
|   | 7.3.1.2 list   | 56                   |
|   | 7.3.1.3 m_rings  | 57                   |
|   | 7.3.1.4 n_threads  | 57                   |
|   |  | 57                   |
|   | 7.3.1.5 next   | 57                   |
|   | 7.3.1.5 next   |                      |
|   | 7.3.1.6 on_closed  | 57                   |
|   | 7.3.1.6 on_closed  | 57<br>57             |
|   | 7.3.1.6 on_closed  | 57<br>57<br>57       |
|   | 7.3.1.6 on_closed  | 57<br>57<br>57<br>58 |

| 7.3.1.11 on_error_arg  | 59 |
|--|----|
| 7.3.1.12 on_readed   | 59 |
| 7.3.1.13 on_readed_arg   | 59 |
| 7.3.1.14 on_wrote  | 59 |
| 7.3.1.15 on_wrote_arg  | 60 |
| 7.3.1.16 pool  | 60 |
| 7.3.1.17 rings   | 60 |
| 7.3.1.18 ss  | 60 |
| 7.3.1.19 status  | 61 |
| 7.3.1.20 status_c  | 61 |
| 7.3.1.21 status_m  | 61 |
| 7.4 low_saurion.h  | 61 |
| 7.5 /w/saurion/saurion/include/low_saurion_secret.h File Reference | 62 |
| 7.6 low_saurion_secret.h   | 62 |
| 7.7 /w/saurion/saurion/include/saurion.hpp File Reference          | 63 |
| 7.8 saurion.hpp  | 63 |
| 7.9 /w/saurion/saurion/include/threadpool.h File Reference         | 64 |
| 7.10 threadpool.h  | 64 |
| 7.11 /w/saurion/saurion/src/linked_list.c File Reference           | 65 |
| 7.11.1 Function Documentation                                      | 65 |
| 7.11.1.1 create_node()   | 66 |
| 7.11.1.2 free_node()   | 66 |
| 7.11.1.3 list_delete_node()  | 67 |
| 7.11.1.4 list_free()   | 67 |
| 7.11.1.5 list_insert()   | 67 |
| 7.11.2 Variable Documentation                                      | 68 |
| 7.11.2.1 list_mutex  | 68 |
| 7.12 linked_list.c   | 68 |
| 7.13 /w/saurion/src/low_saurion.c File Reference                   | 70 |
| 7.13.1 Macro Definition Documentation                              | 72 |
| 7.13.1.1 EV_ACC  | 72 |
| 7.13.1.2 EV_ERR  | 72 |
| 7.13.1.3 EV_REA  | 72 |
| 7.13.1.4 EV_WAI  | 72 |
| 7.13.1.5 EV_WRI  | 72 |
| 7.13.1.6 MAX   | 73 |
| 7.13.1.7 MIN   | 73 |
| 7.13.2 Function Documentation                                      | 73 |
| 7.13.2.1 add_accept()  | 73 |
| 7.13.2.2 add_efd()   | 74 |
| 7.13.2.3 add_read()  | 74 |
| 7.13.2.4 add_read_continue()                                       | 75 |

| 7.13.2.5 add_write()                                    | 75 |
|---|----|
| 7.13.2.6 handle_accept()                                | 76 |
| 7.13.2.7 handle_close()                                 | 76 |
| 7.13.2.8 handle_error()                                 | 77 |
| 7.13.2.9 handle_read()                                  | 77 |
| 7.13.2.10 handle_write()                                | 77 |
| 7.13.2.11 htonll()                                      | 78 |
| 7.13.2.12 next()  | 78 |
| 7.13.2.13 ntohll()                                      | 78 |
| 7.13.2.14 saurion_worker_master()                       | 79 |
| 7.13.2.15 saurion_worker_master_loop_it()               | 79 |
| 7.13.2.16 saurion_worker_slave()                        | 80 |
| 7.13.2.17 saurion_worker_slave_loop_it()                | 81 |
| 7.13.3 Variable Documentation                           | 81 |
| 7.13.3.1 print_mutex                                    | 81 |
| 7.13.3.2 TIMEOUT_RETRY_SPEC                             | 82 |
| 7.14 low_saurion.c                                      | 82 |
| 7.15 /w/saurion/saurion/src/main.c File Reference       | 95 |
| 7.16 main.c   | 95 |
| 7.17 /w/saurion/saurion/src/saurion.cpp File Reference  | 96 |
| 7.18 saurion.cpp  | 96 |
| 7.19 /w/saurion/saurion/src/threadpool.c File Reference | 97 |
| 7.19.1 Macro Definition Documentation                   | 97 |
| 7.19.1.1 FALSE  | 97 |
| 7.19.1.2 TRUE   | 98 |
| 7.19.2 Function Documentation                           | 98 |
| 7.19.2.1 threadpool_worker()                            | 98 |
| 7.20 threadpool.c                                       | 98 |
| Index 1   | 03 |

# Chapter 1

## **Todo List**

```
Member EXTERNAL_set_socket (int p)
Eliminar

Member read_chunk (void **dest, size_t *len, struct request *const req)
add message contraint
validar msg_size, crear maximos
validar offsets
```

2 Todo List

# Chapter 2

# **Module Index**

## 2.1 Modules

Here is a list of all modules:

| LowSaurion | <br> |  |  |  |  |  |  | <br> |  |  |  |  |  |  |  |  |  | <br> |  |  | 9  |
|------------|------|--|--|--|--|--|--|------|--|--|--|--|--|--|--|--|--|------|--|--|----|
| ThreadPool | <br> |  |  |  |  |  |  | <br> |  |  |  |  |  |  |  |  |  | <br> |  |  | 25 |

4 Module Index

# **Chapter 3**

# **Class Index**

## 3.1 Class List

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

| Node .    |   | 31 |
|-----------|---|----|
| request   |   | 32 |
| saurion   |   |    |
|           | Main structure for managing io_uring and socket events          | 34 |
| Saurion   |   | 37 |
| saurion:: | saurion_callbacks   |    |
|           | Structure containing callback functions to handle socket events | 42 |
| saurion_  | callbacks   |    |
|           | Structure containing callback functions to handle socket events | 46 |
| saurion_  | wrapper   | 49 |
| task      |   | 50 |
| threadpo  | ol  | 51 |

6 Class Index

# **Chapter 4**

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

| /w/saurion/saurion/include/linked_list.h        | 53 |
|---|----|
| /w/saurion/saurion/include/low_saurion.h        | 55 |
| /w/saurion/saurion/include/low_saurion_secret.h | 62 |
| /w/saurion/saurion/include/saurion.hpp          | 63 |
| /_w/saurion/saurion/include/threadpool.h        | ô4 |
| /w/saurion/src/linked_list.c                    | ô5 |
| /w/saurion/src/low_saurion.c                    | 70 |
| /_w/saurion/src/main.c                          | 95 |
| /_w/saurion/src/saurion.cpp                     | 96 |
| / w/saurion/saurion/src/threadpool.c            | 97 |

8 File Index

## **Chapter 5**

## **Module Documentation**

## 5.1 LowSaurion

The saurion class is designed to efficiently handle asynchronous input/output events on Linux systems using the io\_uring API. Its main purpose is to manage network operations such as socket connections, reads, writes, and closures by leveraging an event-driven model that enhances performance and scalability in highly concurrent applications.

#### **Classes**

struct saurion

Main structure for managing io\_uring and socket events.

## **Macros**

- #define \_POSIX\_C\_SOURCE 200809L
- #define PACKING\_SZ 32

Defines the memory alignment size for structures in the saurion class.

## **Functions**

- int EXTERNAL\_set\_socket (int p)
- struct saurion \* saurion\_create (uint32\_t n\_threads)

Creates an instance of the saurion structure.

int saurion start (struct saurion \*s)

Starts event processing in the saurion structure.

void saurion\_stop (const struct saurion \*s)

Stops event processing in the saurion structure.

void saurion\_destroy (struct saurion \*s)

Destroys the saurion structure and frees all associated resources.

• void saurion\_send (struct saurion \*s, const int fd, const char \*const msg)

Sends a message through a socket using io\_uring.

- int allocate\_iovec (struct iovec \*iov, size\_t amount, size\_t pos, size\_t size, void \*\*chd\_ptr)
- int initialize\_iovec (struct iovec \*iov, size\_t amount, size\_t pos, const void \*msg, size\_t size, uint8\_t h)

Initializes a specified iovec structure with a message fragment.

• int set\_request (struct request \*\*r, struct Node \*\*I, size\_t s, const void \*m, uint8\_t h)

Sets up a request and allocates iovec structures for data handling in liburing.

int read\_chunk (void \*\*dest, size\_t \*len, struct request \*const req)

Reads a message chunk from the request's iovec buffers, handling messages that may span multiple iovec entries.

void free\_request (struct request \*req, void \*\*children\_ptr, size\_t amount)

## 5.1.1 Detailed Description

The saurion class is designed to efficiently handle asynchronous input/output events on Linux systems using the io\_uring API. Its main purpose is to manage network operations such as socket connections, reads, writes, and closures by leveraging an event-driven model that enhances performance and scalability in highly concurrent applications.

This function allocates memory for each struct iovec

The main structure, saurion, encapsulates io\_uring rings and facilitates synchronization between multiple threads through the use of mutexes and a thread pool that distributes operations in parallel. This allows efficient handling of I/O operations across several sockets simultaneously, without blocking threads during operations.

The messages are composed of three main parts:

- A header, which is an unsigned 64-bit number representing the length of the message body.
- · A body, which contains the actual message data.
- · A footer, which consists of 8 bits set to 0.

For example, for a message with 9000 bytes of content, the header would contain the number 9000, the body would consist of those 9000 bytes, and the footer would be 1 byte set to 0.

When these messages are sent to the kernel, they are divided into chunks using iovec. Each chunk can hold a maximum of 8192 bytes and contains two fields:

- iov\_base, which is an array where the chunk of the message is stored.
- iov\_len, the number of bytes used in the iov\_base array.

For the message with 9000 bytes, the iovec division would look like this:

- The first iovec would contain:
  - 8 bytes for the header (the length of the message body, 9000).
  - 8184 bytes of the message body.
  - iov\_len would be 8192 bytes in total.
- The second iovec would contain:
  - The remaining 816 bytes of the message body.
  - 1 byte for the footer (set to 0).
  - iov\_len would be 817 bytes in total.

The structure of the message is as follows:

| + |           |       | -+- |                | +-  | +        |
|---|-----------|-------|-----|----------------|-----|----------|
|   | Header    |       |     | Body           | - 1 | Footer   |
|   | (64 bits: | 9000) |     | (Message Data) | - 1 | (1 byte) |
| + |           |       | -+- |                | +-  | +        |

The structure of the iovec division is:

Each I/O event can be monitored and managed through custom callbacks that handle connection, read, write, close, or error events on the sockets.

#### Basic usage example:

```
// Create the saurion structure with 4 threads
struct saurion *s = saurion_create(4);
// Start event processing
if (saurion_start(s) != 0) {
    // Handle the error
}
// Send a message through a socket
saurion_send(s, socket_fd, "Hello, World!");
// Stop event processing
saurion_stop(s);
// Destroy the structure and free resources
saurion_destroy(s);
```

In this example, the saurion structure is created with 4 threads to handle the workload. Event processing is started, allowing it to accept connections and manage I/O operations on sockets. After sending a message through a socket, the system can be stopped, and the resources are freed.

## Author

Israel

#### Date

2024

This function allocates memory for each struct iovec. Every struct iovec consists of two member variables:

- iov\_base, a void \* array that will hold the data. All of them will allocate the same amount of memory (CHUNK\_SZ) to avoid memory fragmentation.
- iov\_len, an integer representing the size of the data stored in the iovec. The data size is CHUNK\_SZ unless it's the last one, in which case it will hold the remaining bytes. In addition to initialization, the function adds the pointers to the allocated memory into a child array to simplify memory deallocation later on.

## **Parameters**

| iov     | Structure to initialize.                                       |
|---------|--|
| amount  | Total number of iovec to initialize.                           |
| pos     | Current position of the iovec within the total iovec (amount). |
| size    | Total size of the data to be stored in the iovec.              |
| chd_ptr | Array to hold the pointers to the allocated memory.            |

## **Return values**

| ERROR_CODE   | if there was an error during memory allocation. |
|--------------|---|
| SUCCESS_CODE | if the operation was successful.                |

Note

The last iovec will allocate only the remaining bytes if the total size is not a multiple of CHUNK SZ.

## 5.1.2 Macro Definition Documentation

## 5.1.2.1 POSIX C SOURCE

#define \_POSIX\_C\_SOURCE 200809L

Definition at line 107 of file low\_saurion.h.

## 5.1.2.2 PACKING\_SZ

#define PACKING\_SZ 32

Defines the memory alignment size for structures in the saurion class.

PACKING\_SZ is used to ensure that certain structures, such as saurion\_callbacks, are aligned to a specific memory boundary. This can improve memory access performance and ensure compatibility with certain hardware architectures that require specific alignment.

In this case, the value is set to 32 bytes, meaning that structures marked with  $\__$ attribute $\__$ ((aligned( $\leftarrow$  PACKING\_SZ))) will be aligned to 32-byte boundaries.

Proper alignment can be particularly important in multithreaded environments or when working with low-level system APIs like io\_uring, where unaligned memory accesses may introduce performance penalties.

Adjusting PACKING\_SZ may be necessary depending on the hardware platform or specific performance requirements.

Definition at line 139 of file low\_saurion.h.

## 5.1.3 Function Documentation

## 5.1.3.1 allocate\_iovec()

```
int allocate_iovec (
              struct iovec * iov,
              size_t amount,
              size_t pos,
              size_t size,
              void ** chd_ptr )
Definition at line 159 of file low_saurion.c.
00161 {
       if (!iov || !chd_ptr)
00162
00163
        {
00164
           return ERROR_CODE;
00165
00166
       iov->iov_base = malloc (CHUNK_SZ);
00167
       if (!iov->iov_base)
00168
00169
           return ERROR_CODE;
00170
```

## 5.1.3.2 EXTERNAL\_set\_socket()

if (iov->iov\_len == 0)

00176 chd\_ptr[pos] = iov->iov\_base;
00177 return SUCCESS\_CODE;

iov->iov\_len = CHUNK\_SZ;

```
int EXTERNAL_set_socket ( \inf \ p \ )
```

#### Todo Eliminar

00171

00172

00173

00175

00178 }

## Definition at line 696 of file low\_saurion.c.

```
00697 {
00698
         int sock = 0:
00699
         struct sockaddr_in srv_addr;
00701
         sock = socket (PF_INET, SOCK_STREAM, 0);
00702
         if (sock < 1)
00703
         {
00704
              return ERROR_CODE;
00705
          }
00706
00707
         int enable = 1;
00708
         if (setsockopt (sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof (int)) < 0)</pre>
00709
00710
              return ERROR_CODE;
00711
00712
        memset (&srv_addr, 0, sizeof (srv_addr));
srv_addr.sin_family = AF_INET;
srv_addr.sin_port = htons (p);
srv_addr.sin_addr.s_addr = htonl (INADDR_ANY);
00713
00714
00715
00716
00717
00718
         if (bind (sock, (const struct sockaddr *)&srv_addr, sizeof (srv_addr)) < 0)</pre>
         {
00719
00720
              return ERROR_CODE;
00721
00722
00723
         if (listen (sock, ACCEPT_QUEUE) < 0)</pre>
00724
         {
             return ERROR_CODE;
00726
00727
00728
        return sock;
00729 }
```

iov->iov\_len = (pos == (amount - 1) ? (size % CHUNK\_SZ) : CHUNK\_SZ);

## 5.1.3.3 free\_request()

```
void free_request (
                struct request * req,
                 void ** children_ptr,
                 size_t amount )
Definition at line 91 of file low saurion.c.
00093
         if (children_ptr)
00094
00095
             free (children_ptr);
children_ptr = NULL;
00096
00098
         for (size_t i = 0; i < amount; ++i)</pre>
00099
             free (req->iov[i].iov_base);
req->iov[i].iov_base = NULL;
00100
00101
00102
00103
        free (req);
```

req = NULL; free (children\_ptr);

children\_ptr = NULL;

## 5.1.3.4 initialize\_iovec()

00104 00105 00106

00107 }

Initializes a specified iovec structure with a message fragment.

This function populates the  $iov_base$  of the iovec structure with a portion of the message, depending on the position (pos) in the overall set of iovec structures. The message is divided into chunks, and for the first iovec, a header containing the size of the message is included. Optionally, padding or adjustments can be applied based on the h flag.

## Parameters

| iov  | Pointer to the iovec structure to initialize.   |  |
|--|---|--|
| amount The total number of iovec structures. |   |  |
| pos  | The current position of the iovec within the overall message split.   |  |
| msg  | Pointer to the message to be split across the iovec structures.   |  |
| size   | The total size of the message.  |  |
| h  | A flag (header flag) that indicates whether special handling is needed for the first iovec (adds the message size as a header) or for the last chunk. |  |

## Return values

| SUCCESS_CODE | on successful initialization of the iovec. |
|--------------|--|
| ERROR_CODE   | if the iov or its iov_base is null.        |

#### Note

For the first iovec (when pos == 0), the message size is copied into the beginning of the iov\_base if the header flag (h) is set. Subsequent chunks are filled with message data, and the last chunk may have one byte reduced if h is set.

## Attention

The message must be properly aligned and divided, especially when using the header flag to ensure no memory access issues.

#### Warning

If msg is null, the function will initialize the iov\_base with zeros, essentially resetting the buffer.

## Definition at line 111 of file low\_saurion.c.

```
00113 {
        if (!iov || !iov->iov_base)
00114
00115
         {
00116
             return ERROR CODE:
00117
00118
        if (msg)
00119
         {
             size_t len = iov->iov_len;
char *dest = (char *)iov->iov_base;
char *orig = (char *)msg + pos * CHUNK_SZ;
00120
00121
00122
             size_t cpy_sz = 0;
00123
00124
             if (h)
00125
00126
                  if (pos == 0)
00127
                  {
                     uint64_t send_size = htonll (size);
00128
                      memcpy (dest, &send_size, sizeof (uint64_t));
dest += sizeof (uint64_t);
00129
00130
00131
                      len -= sizeof (uint64_t);
00132
00133
                 else
00134
                  {
                      orig -= sizeof (uint64_t);
00135
00136
00137
                  if ((pos + 1) == amount)
                  --len;
00138
00139
                      cpy_sz = (len < size ? len : size);</pre>
00140
00141
                      dest[cpy_sz] = 0;
00142
00143
00144
            cpy_sz = (len < size ? len : size);</pre>
00145
             memcpy (dest, orig, cpy_sz);
00146
            dest += cpy_sz;
size_t rem = CHUNK_SZ - (dest - (char *)iov->iov_base);
00147
00148
             memset (dest, 0, rem);
00150
        else
00151
00152
             memset ((char *)iov->iov_base, 0, CHUNK_SZ);
00153
00154
        return SUCCESS_CODE;
00155 }
```

## 5.1.3.5 read\_chunk()

Reads a message chunk from the request's iovec buffers, handling messages that may span multiple iovec entries.

This function processes data from a struct request, which contains an array of iovec structures representing buffered data. Each message in the buffers starts with a size\_t value indicating the size of the message, followed by the message content. The function reads the message size, allocates a buffer for the message content, and copies the data from the iovec buffers into this buffer. It handles messages that span multiple iovec entries and manages incomplete messages by storing partial data within the request structure for subsequent reads.

## **Parameters**

| out    | dest | Pointer to a variable where the address of the allocated message buffer will be stored. The buffer is allocated by the function and must be freed by the caller.                                      |
|--------|------|---|
| out    | len  | Pointer to a size_t variable where the length of the read message will be stored. If a complete message is read, *len is set to the message size. If the message is incomplete, *len is set to 0.     |
| in,out | req  | Pointer to a struct request containing the lovec buffers and state information. The function updates the request's state to track the current position within the lovecs and any incomplete messages. |

#### Note

The function assumes that each message is prefixed with its size (of type  $size\_t$ ), and that messages may span multiple iovec entries. It also assumes that the data in the iovec buffers is valid and properly aligned for reading  $size\_t$  values.

## Warning

The caller is responsible for freeing the allocated message buffer pointed to by \*dest when it is no longer needed.

## Returns

int Returns SUCCESS CODE on success, or ERROR CODE on failure (malformed msg).

## **Return values**

| SUCCESS_CODE | No malformed message found. |
|--------------|-----------------------------|
| ERROR_CODE   | Malformed message found.    |

## Todo add message contraint

```
validar msg_size, crear maximos validar offsets
```

## Definition at line 454 of file low saurion.c.

```
00455 {
             (req->iovec_count == 0)
00457
00458
              return ERROR_CODE;
00459
00460
         size_t max_iov_cont = 0; //< Total size of request
for (size_t i = 0; i < req->iovec_count; ++i)
00461
00462
00463
00464
              max_iov_cont += req->iov[i].iov_len;
00465
        size_t cont_sz = 0;
size_t cont_rem = 0;
00466
00467
00468
         size_t curr_iov = 0;
00469
         size_t curr_iov_off = 0;
         size_t dest_off = 0;
void *dest_ptr = NULL;
00470
00471
00472
         if (req->prev && req->prev_size && req->prev_remain)
00473
00474
             cont_sz = req->prev_size;
00475
              cont_rem = req->prev_remain;
```

```
00476
            curr_iov = 0;
00477
             curr_iov_off = 0;
00478
            dest_off = cont_sz - cont_rem;
00479
            if (cont_rem <= max_iov_cont)</pre>
00480
00481
                *dest = req->prev;
00482
                 dest_ptr = *dest;
00483
                 req->prev = NULL;
00484
                 req->prev_size = 0;
00485
                req->prev_remain = 0;
              }
00486
00487
            else
00488
              {
00489
                dest_ptr = req->prev;
00490
                *dest = NULL;
00491
00492
00493
        else if (req->next_iov || req->next_offset)
00494
00495
            curr_iov = req->next_iov;
00496
             curr_iov_off = req->next_offset;
00497
             cont_sz = *(
                (size_t *)(((uint8_t *)req->iov[curr_iov].iov_base) + curr_iov_off));
00498
            cont_sz = ntohll (cont_sz);
curr_iov_off += sizeof (uint64_t);
00499
00500
00501
             cont_rem = cont_sz;
             dest_off = cont_sz - cont_rem;
00502
00503
             if ((curr_iov_off + cont_rem + 1) <= max_iov_cont)</pre>
00504
00505
                 *dest = malloc (cont_sz);
00506
                dest_ptr = *dest;
00507
00508
00509
                req->prev = malloc (cont_sz);
dest_ptr = req->prev;
*dest = NULL;
*len = 0;
00510
00511
00512
00514
00515
00516
        else
00517
        {
            curr_iov = 0;
00518
            curr_iov_off = 0;
00519
00520
            cont_sz = *(
00521
                (size_t *)(((uint8_t *)req->iov[curr_iov].iov_base) + curr_iov_off));
00522
             cont_sz = ntohll (cont_sz);
00523
            curr_iov_off += sizeof (uint64_t);
00524
            cont_rem = cont_sz;
            dest_off = cont_sz - cont_rem;
00525
00526
            if (cont_rem <= max_iov_cont)</pre>
00527
00528
                 *dest = malloc (cont_sz);
00529
                dest_ptr = *dest;
00530
00531
            else
00532
              {
00533
                req->prev = malloc (cont_sz);
00534
                 dest_ptr = req->prev;
00535
                *dest = NULL;
00536
00537
00538
        size_t curr_iov_msq_rem = 0;
00539
00540
        uint8_t ok = 1UL;
00541
        while (1)
00542
00543
            curr iov msg rem
00544
                = MIN (cont_rem, (req->iov[curr_iov].iov_len - curr_iov_off));
            memcpy ((uint8_t *)dest_ptr + dest_off,
00546
                     ((uint8_t *)req->iov[curr_iov].iov_base) + curr_iov_off,
00547
                    curr_iov_msg_rem);
            dest_off += curr_iov_msg_rem;
00548
            curr_iov_off += curr_iov_msg_rem;
00549
00550
            cont_rem -= curr_iov_msg_rem;
00551
            if (cont_rem <= 0)
00552
              {
00553
                 if (*(((uint8_t *)req->iov[curr_iov].iov_base) + curr_iov_off) != 0)
00554
00555
                    ok = OUI:
00556
                 *len = cont_sz;
00558
                 ++curr_iov_off;
00559
                 break;
00560
             if (curr_iov_off >= (req->iov[curr_iov].iov_len))
00561
00562
```

```
++curr_iov;
00564
                 if (curr_iov == req->iovec_count)
00565
00566
                     break;
00567
00568
                 curr_iov_off = 0;
00569
00570
          }
00571
00572
        if (req->prev)
00573
         {
            req->prev_size = cont_sz;
req->prev_remain = cont_rem;
00574
00575
00576
             *dest = NULL;
00577
             len = 0;
00578
00579
        else
00580
         {
00581
            req->prev_size = 0;
00582
            req->prev_remain = 0;
00583
00584
        if (curr_iov < req->iovec_count)
00585
         {
00586
             \label{eq:continuous} \mbox{uint64\_t next\_sz = *(uint64\_t *)(((uint8\_t *)req->iov[curr_iov].iov\_base))} \\
00587
                                                 + curr_iov_off);
00588
             if ((req->iov[curr_iov].iov_len > curr_iov_off) && next_sz)
00589
00590
                 req->next_iov = curr_iov;
00591
                 req->next_offset = curr_iov_off;
00592
00593
             else
00594
             {
00595
               req->next_iov = 0;
00596
                 req->next_offset = 0;
00597
00598
          }
00599
00600
        if (ok)
00601
        {
00602
            return SUCCESS_CODE;
00603
00604
        free (dest_ptr);
        dest_ptr = NULL;
*dest = NULL;
*len = 0;
00605
00606
00607
00608
        req->next_iov = 0;
00609
        req->next_offset = 0;
00610
        for (size_t i = curr_iov; i < req->iovec_count; ++i)
00611
00612
             for (size_t j = curr_iov_off; j < req->iov[i].iov_len; ++j)
00613
00614
                uint8_t foot = *((uint8_t *)req->iov[i].iov_base) + j;
00615
                 if (foot == 0)
00616
                     req->next_iov = i;
req->next_offset = (j + 1) % req->iov[i].iov_len;
00617
00618
                     return ERROR_CODE;
00620
00621
00622
00623 return ERROR CODE;
00624 }
```

## 5.1.3.6 saurion\_create()

Creates an instance of the saurion structure.

This function initializes the saurion structure, sets up the eventfd, and configures the io\_uring queue, preparing it for use. It also sets up the thread pool and any necessary synchronization mechanisms.

#### **Parameters**

*n threads* The number of threads to initialize in the thread pool.

#### Returns

struct saurion\* A pointer to the newly created saurion structure, or NULL if an error occurs.

## Definition at line 733 of file low\_saurion.c.

```
LOG_INIT (" ");
00735
00736
        struct saurion *p = (struct saurion *)malloc (sizeof (struct saurion));
00737
        if (!p)
00738
         {
00739
           LOG_END (" ");
00740
            return NULL;
00741
        int ret = 0;
00742
00743
        ret = pthread_mutex_init (&p->status_m, NULL);
00744
        if (ret)
00745
         {
            free (p);
LOG_END (" ");
00746
00747
00748
           return NULL;
00749
00750
        ret = pthread cond init (&p->status c, NULL);
00751
        if (ret)
00752
         {
            free (p);
LOG_END (" ");
00753
00754
00755
            return NULL;
00756
00757
        p->m rings
00758
            = (pthread_mutex_t *)malloc (n_threads * sizeof (pthread_mutex_t));
00759
        if (!p->m_rings)
00760
           free (p);
LOG_END (" ");
00761
00762
00763
            return NULL;
00764
00765
        for (uint32_t i = 0; i < n_threads; ++i)</pre>
00766
00767
            pthread_mutex_init (&(p->m_rings[i]), NULL);
00768
00769
        p->ss=0;
00770
        n_threads = (n_threads < 2 ? 2 : n_threads);</pre>
00771
        n_threads = (n_threads > NUM_CORES ? NUM_CORES : n_threads);
00772
        p->n_threads = n_threads;
00773
        p->status = 0;
00774
        p->list = NULL;
00775
        p->cb.on connected = NULL;
00776
        p->cb.on_connected_arg = NULL;
00777
        p->cb.on_readed = NULL;
00778
       p->cb.on_readed_arg = NULL;
00779
        p->cb.on_wrote = NULL;
00780
        p->cb.on_wrote_arg = NULL;
00781
       p->cb.on_closed = NULL;
00782
        p->cb.on_closed_arg = NULL;
        p->cb.on_error = NULL;
00783
00784
        p->cb.on_error_arg = NULL;
        p->next = 0;
p->efds = (int *)malloc (sizeof (int) * p->n_threads);
00785
00786
00787
        if (!p->efds)
00788
          {
00789
            free (p->m_rings);
            free (p);
LOG_END (" ");
00790
00791
00792
            return NULL;
00793
        for (uint32_t i = 0; i < p->n_threads; ++i)
00794
00795
00796
            p->efds[i] = eventfd (0, EFD_NONBLOCK);
00797
            if (p->efds[i] == ERROR_CODE)
00798
00799
                for (uint32_t j = 0; j < i; ++j)
00800
00801
                    close (p->efds[i]);
00802
00803
                free (p->efds);
00804
                free (p->m_rings);
```

```
free (p);
LOG_END (" ");
00805
00806
00807
                return NULL;
00808
00809
00810
        p->rings
            = (struct io_uring *)malloc (sizeof (struct io_uring) * p->n_threads);
00811
00812
        if (!p->rings)
00813
00814
            for (uint32_t j = 0; j < p->n_threads; ++j)
00815
              {
00816
                close (p->efds[j]);
00817
00818
            free (p->efds);
00819
            free (p->m_rings);
            free (p);
LOG_END (" ");
00820
00821
00822
            return NULL;
00823
00824
        for (uint32_t i = 0; i < p->n_threads; ++i)
00825
00826
            memset (&p->rings[i], 0, sizeof (struct io_uring));
            ret = io_uring_queue_init (SAURION_RING_SIZE, &p->rings[i], 0);
00827
00828
            if (ret)
00829
              {
00830
                for (uint32_t j = 0; j < p->n_threads; ++j)
00831
00832
                    close (p->efds[j]);
00833
00834
                free (p->efds);
00835
                free (p->rings);
00836
                free (p->m_rings);
                free (p);
LOG_END (" ");
00837
00838
00839
                return NULL;
00840
00841
00842
       p->pool = threadpool_create (p->n_threads);
00843
        LOG_END (" ");
00844
        return p;
00845 }
```

## 5.1.3.7 saurion\_destroy()

```
void saurion_destroy (
          struct saurion * s )
```

Destroys the saurion structure and frees all associated resources.

This function waits for the event processing to stop, frees the memory used by the saurion structure, and closes any open file descriptors. It ensures that no resources are leaked when the structure is no longer needed.

#### **Parameters**

s | Pointer to the saurion structure.

## Definition at line 1085 of file low\_saurion.c.

```
01086 {
01087
        pthread mutex lock (&s->status m);
01088
        while (s->status > 0)
01089
01090
             pthread_cond_wait (&s->status_c, &s->status_m);
01091
01092
        pthread_mutex_unlock (&s->status_m);
01093
        threadpool_destroy (s->pool);
for (uint32_t i = 0; i < s->n_threads; ++i)
01094
01095
01096
             io_uring_queue_exit (&s->rings[i]);
01097
             pthread_mutex_destroy (&s->m_rings[i]);
01098
01099
        free (s->m_rings);
```

```
list_free (&s->list);
01101
       for (uint32_t i = 0; i < s->n_threads; ++i)
01102
01103
           close (s->efds[i]);
01104
       free (s->efds);
01105
       if (!s->ss)
01106
01107
        {
01108
           close (s->ss);
01109
       free (s->rings);
01110
01111
       pthread_mutex_destroy (&s->status_m);
       pthread_cond_destroy (&s->status_c);
01112
01113
       free (s);
01114 }
```

## 5.1.3.8 saurion\_send()

```
void saurion_send (  struct \ saurion * s, \\ const \ int \ fd, \\ const \ char *const \ msg \ )
```

Sends a message through a socket using io\_uring.

This function prepares and sends a message through the specified socket using the io\_uring event queue. The message is split into iovec structures for efficient transmission and sent asynchronously.

#### **Parameters**

| s Pointer to the saurion structure.                       |  |  |
|---|--|--|
| fd  | File descriptor of the socket to which the message will be ser |  |
| msg Pointer to the character string (message) to be sent. |  |  |

## Definition at line 1117 of file low\_saurion.c.

```
01118 {
01119 add_write (s, fd, msg, next (s));
01120 }
```

## 5.1.3.9 saurion start()

```
int saurion_start ( struct \ saurion * s \ )
```

Starts event processing in the saurion structure.

This function begins accepting socket connections and handling io\_uring events in a loop. It will run continuously until a stop signal is received, allowing the application to manage multiple socket events asynchronously.

#### **Parameters**

```
s | Pointer to the saurion structure.
```

#### Returns

int Returns 0 on success, or 1 if an error occurs.

Definition at line 1044 of file low\_saurion.c.

```
01045 {
        pthread_mutex_init (&print_mutex, NULL);
01046
        threadpool_init (s->pool);
threadpool_add (s->pool, saurion_worker_master, s);
01047
01048
01049
        struct saurion_wrapper *ss = NULL;
01050
        for (uint32_t i = 1; i < s->n_threads; ++i)
01051
01052
            ss = (struct saurion_wrapper *)malloc (sizeof (struct saurion_wrapper));
01053
            if (!ss)
01054
              {
01055
                return ERROR_CODE;
             }
01056
            ss->s = s;
ss->sel = i;
01057
01058
01059
            threadpool_add (s->pool, saurion_worker_slave, ss);
01060
01061
        pthread_mutex_lock (&s->status_m);
01062
        while (s->status < (int)s->n_threads)
01063
            pthread_cond_wait (&s->status_c, &s->status_m);
01064
01065
01066
       pthread_mutex_unlock (&s->status_m);
01067
        return SUCCESS_CODE;
01068 }
```

## 5.1.3.10 saurion\_stop()

```
void saurion_stop ( {\tt const\ struct\ saurion} \ *\ s\ )
```

Stops event processing in the saurion structure.

This function sends a signal to the eventfd, indicating that the event loop should stop. It gracefully shuts down the processing of any remaining events before exiting.

#### **Parameters**

```
s | Pointer to the saurion structure.
```

Definition at line 1071 of file low\_saurion.c.

```
01073
        uint64_t u = 1;
        for (uint32_t i = 0; i < s->n_threads; ++i)
01074
01075
01076
            while (write (s->efds[i], &u, sizeof (u)) < 0)</pre>
01077
              {
01078
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
01079
01080
01081
        threadpool_wait_empty (s->pool);
01082 }
```

## 5.1.3.11 set\_request()

```
struct Node ** 1,
size_t s,
const void * m,
uint8_t h ) [private]
```

Sets up a request and allocates iovec structures for data handling in liburing.

This function configures a request structure that will be used to send or receive data through liburing's submission queues. It allocates the necessary iovec structures to split the data into manageable chunks, and optionally adds a header if specified. The request is inserted into a list tracking active requests for proper memory management and deallocation upon completion.

#### **Parameters**

| r | Pointer to a pointer to the request structure. If NULL, a new request is created.      |  |
|---|--|--|
| 1 | Pointer to the list of active requests (Node list) where the request will be inserted. |  |
| s | Size of the data to be handled. Adjusted if the header flag (h) is true.               |  |
| m | Pointer to the memory block containing the data to be processed.                       |  |
| h | Header flag. If true, a header (sizeof(uint64_t) + 1) is added to the iovec data.      |  |

#### Returns

int Returns SUCCESS\_CODE on success, or ERROR\_CODE on failure (memory allocation issues or insertion failure).

#### Return values

| SUCCESS_CODE | The request was successfully set up and inserted into the list.                      |
|--------------|--|
| ERROR_CODE   | Memory allocation failed, or there was an error inserting the request into the list. |

#### Note

The function handles memory allocation for the request and iovec structures, and ensures that the memory is freed properly if an error occurs. Pointers to the iovec blocks (children\_ptr) are managed and used for proper memory deallocation.

## Definition at line 182 of file low saurion.c.

```
00184 {
00185
         uint64_t full_size = s;
00186
         if (h)
00187
             full_size += (sizeof (uint64_t) + sizeof (uint8_t));
00189
         size_t amount = full_size / CHUNK_SZ;
00190
         amount = amount + (full_size % CHUNK_SZ == 0 ? 0 : 1);
struct request *temp = (struct request *)malloc (
00191
00192
00193
             sizeof (struct request) + sizeof (struct iovec) * amount);
00194
         if (!temp)
00195
           {
00196
             return ERROR_CODE;
00197
00198
         if (!*r)
00199
00200
              *r = temp;
00201
             (*r)->prev = NULL;
00202
              (*r)->prev_size = 0;
00203
             (*r) \rightarrow prev\_remain = 0;
00204
              (*r) \rightarrow next_{iov} = 0;
00205
              (*r)->next_offset = 0;
00206
00207
        else
```

5.2 ThreadPool 25

```
00209
            temp->client_socket = (*r)->client_socket;
00210
            temp->event_type = (*r)->event_type;
            temp->prev = (*r)->prev;
00211
            temp->prev_size = (*r)->prev_size;
temp->prev_remain = (*r)->prev_remain;
00212
00213
            temp->next_iov = (*r)->next_iov;
00215
            temp->next_offset = (*r)->next_offset;
00216
            *r = temp;
00217
00218
       struct request *req = *r;
00219
        reg->iovec count = (int)amount;
00220
        void **children_ptr = (void **)malloc (amount * sizeof (void *));
00221
       if (!children_ptr)
00222
        {
00223
            free_request (req, children_ptr, 0);
00224
            return ERROR CODE;
00225
        for (size_t i = 0; i < amount; ++i)</pre>
00228
            if (!allocate_iovec (&req->iov[i], amount, i, full_size, children_ptr))
00229
00230
                free_request (req, children_ptr, amount);
00231
                return ERROR CODE;
00232
00233
            if (!initialize_iovec (&req->iov[i], amount, i, m, s, h))
00234
              {
00235
                free_request (req, children_ptr, amount);
00236
                return ERROR_CODE;
00237
00238
00239
       if (list_insert (1, req, amount, children_ptr))
00240
00241
            free_request (req, children_ptr, amount);
00242
            return ERROR_CODE;
00243
00244
       free (children_ptr);
00245
       return SUCCESS_CODE;
00246 }
```

## 5.2 ThreadPool

## **Functions**

- struct threadpool \* threadpool\_create (size\_t num\_threads)
- struct threadpool \* threadpool\_create\_default (void)
- void threadpool\_init (struct threadpool \*pool)
- void threadpool\_add (struct threadpool \*pool, void(\*function)(void \*), void \*argument)
- void threadpool\_stop (struct threadpool \*pool)
- int threadpool\_empty (struct threadpool \*pool)
- void threadpool\_wait\_empty (struct threadpool \*pool)
- void threadpool\_destroy (struct threadpool \*pool)

## 5.2.1 Detailed Description

## 5.2.2 Function Documentation

## 5.2.2.1 threadpool\_add()

```
void threadpool_add (
               struct threadpool * pool,
               void(*)(void *) function,
               void * argument )
Definition at line 175 of file threadpool.c.
00178
        LOG_INIT (" ");
        if (pool == NULL || function == NULL)
00179
00180
            LOG_END (" ");
00181
00182
            return;
00183
         }
00184
00185
        struct task *new_task = malloc (sizeof (struct task));
00186
        if (new_task == NULL)
00187
            perror ("Failed to allocate task");
00188
            LOG_END (" ");
00189
00190
            return;
00191
00192
        new_task->function = function;
new_task->argument = argument;
00193
00194
        new_task->next = NULL;
00195
00196
00197
        pthread_mutex_lock (&pool->queue_lock);
00198
00199
        if (pool->task_queue_head == NULL)
00200
00201
            pool->task queue head = new task;
            pool->task_queue_tail = new_task;
00202
00203
00204
00205
            pool->task_queue_tail->next = new_task;
00206
00207
            pool->task_queue_tail = new_task;
00208
        pthread_cond_signal (&pool->queue_cond);
00210
00211
        pthread_mutex_unlock (&pool->queue_lock);
00212
        LOG_END (" ");
00213 }
```

## 5.2.2.2 threadpool\_create()

## Definition at line 32 of file threadpool.c.

```
00033 {
00034
        LOG_INIT (" ");
        struct threadpool *pool = malloc (sizeof (struct threadpool));
00036
        if (pool == NULL)
00037
            perror ("Failed to allocate threadpool"); LOG_END (" ");
00038
00039
            return NULL;
00040
00041
00042
        if (num_threads < 3)</pre>
00043
00044
            num_threads = 3;
00045
00046
        if (num_threads > NUM_CORES)
00047
        {
00048
            num_threads = NUM_CORES;
00049
00050
00051
        pool->num_threads = num_threads;
00052
        pool->threads = malloc (sizeof (pthread_t) * num_threads);
00053
        if (pool->threads == NULL)
00054
          {
```

5.2 ThreadPool 27

```
perror ("Failed to allocate threads array");
             free (pool);
LOG_END (" ");
00056
00057
             return NULL;
00058
00059
00060
00061
        pool->task_queue_head = NULL;
00062
        pool->task_queue_tail = NULL;
00063
        pool->stop = FALSE;
00064
        pool->started = FALSE;
00065
00066
        if (pthread_mutex_init (&pool->queue_lock, NULL) != 0)
00067
          {
00068
            perror ("Failed to initialize mutex");
00069
             free (pool->threads);
            free (pool);
LOG_END (" ");
00070
00071
00072
            return NULL;
00073
00074
00075
        if (pthread_cond_init (&pool->queue_cond, NULL) != 0)
00076
00077
            perror ("Failed to initialize condition variable");
00078
             pthread_mutex_destroy (&pool->queue_lock);
free (pool->threads);
00079
00080
             free (pool);
LOG_END (" ");
00081
00082
             return NULL;
00083
00084
00085
        if (pthread_cond_init (&pool->empty_cond, NULL) != 0)
00086
00087
            perror ("Failed to initialize empty condition variable");
00088
             pthread_mutex_destroy (&pool->queue_lock);
00089
             pthread_cond_destroy (&pool->queue_cond);
            free (pool->threads);
free (pool);
LOG_END (" ");
00090
00091
00092
00093
             return NULL;
00094
00095
       LOG_END (" ");
00096
00097
        return pool;
00098 }
```

## 5.2.2.3 threadpool\_create\_default()

#### Definition at line 101 of file threadpool.c.

```
00102 {
00103    return threadpool_create (NUM_CORES);
00104 }
```

## 5.2.2.4 threadpool\_destroy()

```
void threadpool_destroy ( {\tt struct\ threadpool\ *\ pool\ )}
```

## Definition at line 274 of file threadpool.c.

```
00275 {
00276   LOG_INIT (" ");
00277   if (pool == NULL)
00278   {
00279    LOG_END (" ");
00280    return;
00281  }
00282   threadpool_stop (pool);
```

```
00284
          pthread_mutex_lock (&pool->queue_lock);
          struct task *task = pool->task_queue_head;
while (task != NULL)
00285
00286
00287
               struct task *tmp = task;
00288
                task = task->next;
00290
               free (tmp);
00291
00292
          pthread_mutex_unlock (&pool->queue_lock);
00293
         pthread_mutex_destroy (&pool->queue_lock);
pthread_cond_destroy (&pool->queue_cond);
pthread_cond_destroy (&pool->empty_cond);
00294
00295
00296
00297
00298
          free (pool->threads);
         free (pool);
LOG_END (" ");
00299
00300
00301 }
```

## 5.2.2.5 threadpool\_empty()

#### Definition at line 240 of file threadpool.c.

```
00241 {
00242
        LOG_INIT (" ");
        if (pool == NULL)
00243
00244
            LOG_END (" ");
00245
            return TRUE;
00246
00247
00248 pthread_mutex_lock (&pool->queue_lock);
00249
        int empty = (pool->task_queue_head == NULL);
       pthread_mutex_unlock (&pool->queue_lock);
LOG_END (" ");
00250
00251
00252
       return empty;
00253 }
```

## 5.2.2.6 threadpool\_init()

## Definition at line 151 of file threadpool.c.

```
00153
       LOG_INIT (" ");
       if (pool == NULL || pool->started)
00154
00155
00156
           LOG_END (" ");
           return;
00158
00159
       for (size_t i = 0; i < pool->num_threads; i++)
00160
00161
           if (pthread_create (&pool->threads[i], NULL, threadpool_worker,
00162
                                (void *)pool)
00163
               != 0)
00164
00165
               perror ("Failed to create thread");
00166
               pool->stop = TRUE;
00167
               break;
00168
             }
00169
       pool->started = TRUE;
00170
00171 LOG_END (" ");
00172 }
```

5.2 ThreadPool 29

#### 5.2.2.7 threadpool\_stop()

```
void threadpool_stop (
              struct threadpool * pool )
Definition at line 216 of file threadpool.c.
00217 +
00218
        LOG_INIT (" ");
00219
        if (pool == NULL || !pool->started)
00220
            LOG_END (" ");
00221
00222
            return;
00223
00224
        threadpool_wait_empty (pool);
00225
00226
        pthread_mutex_lock (&pool->queue_lock);
00227
        pool->stop = TRUE;
        pthread_cond_broadcast (&pool->queue_cond);
00228
00229
        pthread_mutex_unlock (&pool->queue_lock);
00230
00231
        for (size_t i = 0; i < pool->num_threads; i++)
00232
00233
            pthread_join (pool->threads[i], NULL);
00234
        pool->started = FALSE;
LOG_END (" ");
00235
00236
00237 }
```

#### 5.2.2.8 threadpool wait empty()

```
void threadpool_wait_empty ( {\tt struct\ threadpool\ *\ pool\ )}
```

#### Definition at line 256 of file threadpool.c.

```
00257 {
00258
         LOG_INIT (" ");
         if (pool == NULL)
00259
00260
00261
              LOG_END (" ");
00262
              return;
00263
         pthread_mutex_lock (&pool->queue_lock);
while (pool->task_queue_head != NULL)
00264
00265
00266
00267
              pthread_cond_wait (&pool->empty_cond, &pool->queue_lock);
00268
         pthread_mutex_unlock (&pool->queue_lock);
LOG_END (" ");
00269
00270
00271 }
```

30 Module Documentation

# **Chapter 6**

# **Class Documentation**

## 6.1 Node Struct Reference

Collaboration diagram for Node:

#### **Public Attributes**

- void \* ptr
- size\_t size
- struct Node \*\* children
- struct Node \* next

# 6.1.1 Detailed Description

Definition at line 6 of file linked\_list.c.

## 6.1.2 Member Data Documentation

## 6.1.2.1 children

```
struct Node** Node::children
```

Definition at line 10 of file linked\_list.c.

## 6.1.2.2 next

```
struct Node* Node::next
```

Definition at line 11 of file linked\_list.c.

#### 6.1.2.3 ptr

```
void* Node::ptr
```

Definition at line 8 of file linked\_list.c.

#### 6.1.2.4 size

```
size_t Node::size
```

Definition at line 9 of file linked\_list.c.

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

• /\_\_w/saurion/saurion/src/linked\_list.c

# 6.2 request Struct Reference

## **Public Attributes**

- void \* prev
- size\_t prev\_size
- size\_t prev\_remain
- size\_t next\_iov
- size\_t next\_offset
- int event\_type
- size\_t iovec\_count
- int client\_socket
- struct iovec iov []

## 6.2.1 Detailed Description

Definition at line 32 of file low\_saurion.c.

## 6.2.2 Member Data Documentation

#### 6.2.2.1 client\_socket

```
int request::client_socket
```

Definition at line 41 of file low\_saurion.c.

## 6.2.2.2 event\_type

```
\verb|int request::event_type|\\
```

Definition at line 39 of file low\_saurion.c.

#### 6.2.2.3 iov

```
struct iovec request::iov[]
```

Definition at line 42 of file low\_saurion.c.

## 6.2.2.4 iovec\_count

```
size_t request::iovec_count
```

Definition at line 40 of file low\_saurion.c.

## 6.2.2.5 next\_iov

```
size_t request::next_iov
```

Definition at line 37 of file low\_saurion.c.

#### 6.2.2.6 next offset

```
size_t request::next_offset
```

Definition at line 38 of file low\_saurion.c.

## 6.2.2.7 prev

void\* request::prev

Definition at line 34 of file low\_saurion.c.

#### 6.2.2.8 prev\_remain

```
size_t request::prev_remain
```

Definition at line 36 of file low\_saurion.c.

#### 6.2.2.9 prev\_size

```
size_t request::prev_size
```

Definition at line 35 of file low\_saurion.c.

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

/ w/saurion/saurion/src/low saurion.c

## 6.3 saurion Struct Reference

Main structure for managing io\_uring and socket events.

```
#include <low_saurion.h>
```

Collaboration diagram for saurion:

## **Classes**

· struct saurion callbacks

Structure containing callback functions to handle socket events.

## **Public Attributes**

- struct io\_uring \* rings
- pthread\_mutex\_t \* m\_rings
- int ss
- int \* efds
- struct Node \* list
- pthread\_mutex\_t status\_m
- pthread\_cond\_t status\_c
- int status
- struct threadpool \* pool
- uint32\_t n\_threads
- · uint32\_t next

## 6.3.1 Detailed Description

Main structure for managing io\_uring and socket events.

This structure contains all the necessary data to handle the io\_uring event queue and the callbacks for socket events, enabling efficient asynchronous I/O operations.

Definition at line 148 of file low\_saurion.h.

#### 6.3.2 Member Data Documentation

#### 6.3.2.1 efds

int\* saurion::efds

Eventfd descriptors used for internal signaling between threads.

Definition at line 157 of file low\_saurion.h.

#### 6.3.2.2 list

```
struct Node* saurion::list
```

Linked list for storing active requests.

Definition at line 159 of file low\_saurion.h.

## 6.3.2.3 m\_rings

```
pthread_mutex_t* saurion::m_rings
```

Array of mutexes to protect the io\_uring rings.

Definition at line 153 of file low\_saurion.h.

#### 6.3.2.4 n\_threads

```
uint32_t saurion::n_threads
```

Number of threads in the thread pool.

Definition at line 169 of file low\_saurion.h.

## 6.3.2.5 next

```
uint32_t saurion::next
```

Index of the next io\_uring ring to which an event will be added.

Definition at line 171 of file low\_saurion.h.

## 6.3.2.6 pool

```
struct threadpool* saurion::pool
```

Thread pool for executing tasks in parallel.

Definition at line 167 of file low\_saurion.h.

#### 6.3.2.7 rings

```
struct io_uring* saurion::rings
```

Array of io\_uring structures for managing the event queue.

Definition at line 151 of file low\_saurion.h.

## 6.3.2.8 ss

int saurion::ss

Server socket descriptor for accepting connections.

Definition at line 155 of file low\_saurion.h.

#### 6.3.2.9 status

int saurion::status

Current status of the structure (e.g., running, stopped).

Definition at line 165 of file low\_saurion.h.

#### 6.3.2.10 status\_c

```
pthread_cond_t saurion::status_c
```

Condition variable to signal changes in the structure's state.

Definition at line 163 of file low\_saurion.h.

#### 6.3.2.11 status\_m

```
pthread_mutex_t saurion::status_m
```

Mutex to protect the state of the structure.

Definition at line 161 of file low\_saurion.h.

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

• /\_\_w/saurion/saurion/include/low\_saurion.h

## 6.4 Saurion Class Reference

```
#include <saurion.hpp>
```

Collaboration diagram for Saurion:

#### **Public Types**

- using ConnectedCb = void(\*)(const int, void \*)
- using ReadedCb = void(\*)(const int, const void \*const, const ssize\_t, void \*)
- using WroteCb = void(\*)(const int, void \*)
- using ClosedCb = void(\*)(const int, void \*)
- using ErrorCb = void(\*)(const int, const char \*const, const ssize\_t, void \*)

## **Public Member Functions**

- Saurion (const uint32\_t thds, const int sck) noexcept
- ∼Saurion ()
- Saurion (const Saurion &)=delete
- Saurion (Saurion &&)=delete
- Saurion & operator= (const Saurion &)=delete
- Saurion & operator= (Saurion &&)=delete
- void init () noexcept
- void stop () const noexcept
- Saurion \* on\_connected (ConnectedCb ncb, void \*arg) noexcept
- Saurion \* on\_readed (ReadedCb ncb, void \*arg) noexcept
- Saurion \* on\_wrote (WroteCb ncb, void \*arg) noexcept
- Saurion \* on\_closed (ClosedCb ncb, void \*arg) noexcept
- Saurion \* on\_error (ErrorCb ncb, void \*arg) noexcept
- void send (const int fd, const char \*const msg) noexcept

## **Private Attributes**

• struct saurion \* s

## 6.4.1 Detailed Description

Definition at line 7 of file saurion.hpp.

## 6.4.2 Member Typedef Documentation

## 6.4.2.1 ClosedCb

```
using Saurion::ClosedCb = void (*) (const int, void *)
```

Definition at line 14 of file saurion.hpp.

#### 6.4.2.2 ConnectedCb

```
using Saurion::ConnectedCb = void (*) (const int, void *)
```

Definition at line 10 of file saurion.hpp.

#### 6.4.2.3 ErrorCb

```
using Saurion::ErrorCb = void (*) (const int, const char *const, const ssize_t, void *)
```

Definition at line 15 of file saurion.hpp.

#### 6.4.2.4 ReadedCb

```
using Saurion::ReadedCb = void (*) (const int, const void *const, const ssize_t, void *)
```

Definition at line 11 of file saurion.hpp.

## 6.4.2.5 WroteCb

```
using Saurion::WroteCb = void (*) (const int, void *)
```

Definition at line 13 of file saurion.hpp.

## 6.4.3 Constructor & Destructor Documentation

## 6.4.3.1 Saurion() [1/3]

#### Definition at line 5 of file saurion.cpp.

```
00006 {
00007    this->s = saurion_create (thds);
00008    if (!this->s)
00009     {
00010         return;
00011    }
00012    this->s->ss = sck;
00013 }
```

#### 6.4.3.2 ~Saurion()

```
Saurion::\simSaurion ( )
```

## Definition at line 15 of file saurion.cpp.

```
00015 { saurion_destroy (this->s); }
```

## 6.4.3.3 Saurion() [2/3]

## 6.4.3.4 Saurion() [3/3]

## 6.4.4 Member Function Documentation

## 6.4.4.1 init()

```
void Saurion::init ( ) [noexcept]

Definition at line 18 of file saurion.cpp.

00019 {
00020         if (!saurion_start (this->s))
00021         {
00022             return;
00023         }
00024 }
```

#### 6.4.4.2 on\_closed()

## Definition at line 57 of file saurion.cpp.

## 6.4.4.3 on\_connected()

## Definition at line 33 of file saurion.cpp.

#### 6.4.4.4 on\_error()

#### Definition at line 65 of file saurion.cpp.

```
00066 {
00067     s->cb.on_error = ncb;
00068     s->cb.on_error_arg = arg;
00069     return this;
00070 }
```

#### 6.4.4.5 on\_readed()

#### Definition at line 41 of file saurion.cpp.

## 6.4.4.6 on\_wrote()

#### Definition at line 49 of file saurion.cpp.

## 6.4.4.7 operator=() [1/2]

#### 6.4.4.8 operator=() [2/2]

#### 6.4.4.9 send()

#### Definition at line 73 of file saurion.cpp.

```
00074 {
00075    saurion_send (this->s, fd, msg);
00076 }
```

#### 6.4.4.10 stop()

```
void Saurion::stop ( ) const [noexcept]

Definition at line 27 of file saurion.cpp.
00028 {
00029     saurion_stop (this->s);
00030 }
```

#### 6.4.5 Member Data Documentation

#### 6.4.5.1 s

```
struct saurion* Saurion::s [private]
```

Definition at line 38 of file saurion.hpp.

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

- /\_\_w/saurion/saurion/include/saurion.hpp
- /\_\_w/saurion/saurion/src/saurion.cpp

# 6.5 saurion::saurion\_callbacks Struct Reference

Structure containing callback functions to handle socket events.

```
#include <low_saurion.h>
```

## **Public Attributes**

```
    void(* on_connected )(const int fd, void *arg)
```

Callback for handling new connections.

- void \* on\_connected\_arg
- void(\* on\_readed )(const int fd, const void \*const content, const ssize\_t len, void \*arg)

Callback for handling read events.

- void \* on readed arg
- void(\* on\_wrote )(const int fd, void \*arg)

Callback for handling write events.

- void \* on\_wrote\_arg
- void(\* on\_closed )(const int fd, void \*arg)

Callback for handling socket closures.

- void \* on\_closed\_arg
- void(\* on\_error )(const int fd, const char \*const content, const ssize\_t len, void \*arg)

Callback for handling error events.

void \* on\_error\_arg

## 6.5.1 Detailed Description

Structure containing callback functions to handle socket events.

This structure holds pointers to callback functions for handling events such as connection establishment, reading, writing, closing, and errors on sockets. Each callback has an associated argument pointer that can be passed along when the callback is invoked.

Definition at line 181 of file low\_saurion.h.

#### 6.5.2 Member Data Documentation

### 6.5.2.1 on\_closed

```
void(* saurion::saurion_callbacks::on_closed) (const int fd, void *arg)
```

Callback for handling socket closures.

#### **Parameters**

| fd  | File descriptor of the closed socket. |
|-----|---------------------------------------|
| arg | Additional user-provided argument.    |

Definition at line 221 of file low\_saurion.h.

## 6.5.2.2 on\_closed\_arg

void\* saurion::saurion\_callbacks::on\_closed\_arg

Additional argument for the close callback.

Definition at line 223 of file low\_saurion.h.

## 6.5.2.3 on\_connected

void(\* saurion::saurion\_callbacks::on\_connected) (const int fd, void \*arg)

Callback for handling new connections.

### **Parameters**

| fd  | File descriptor of the connected socket. |
|-----|--|
| arg | Additional user-provided argument.       |

Generated by Doxygen

Definition at line 189 of file low\_saurion.h.

#### 6.5.2.4 on\_connected\_arg

void\* saurion::saurion\_callbacks::on\_connected\_arg

Additional argument for the connection callback.

Definition at line 191 of file low\_saurion.h.

#### 6.5.2.5 on\_error

void(\* saurion::saurion\_callbacks::on\_error) (const int fd, const char \*const content, const
ssize\_t len, void \*arg)

Callback for handling error events.

#### **Parameters**

| fd      | File descriptor of the socket where the error occurred. |
|---------|---|
| content | Pointer to the error message.                           |
| len     | Length of the error message.                            |
| arg     | Additional user-provided argument.                      |

Definition at line 233 of file low saurion.h.

#### 6.5.2.6 on\_error\_arg

void\* saurion::saurion\_callbacks::on\_error\_arg

Additional argument for the error callback.

Definition at line 236 of file low\_saurion.h.

## 6.5.2.7 on\_readed

void(\* saurion::saurion\_callbacks::on\_readed) (const int fd, const void \*const content, const
ssize\_t len, void \*arg)

Callback for handling read events.

#### **Parameters**

| fd      | File descriptor of the socket.     |
|---------|------------------------------------|
| content | Pointer to the data that was read. |
| len     | Length of the data that was read.  |
| arg     | Additional user-provided argument. |

Definition at line 201 of file low\_saurion.h.

## 6.5.2.8 on\_readed\_arg

void\* saurion::saurion\_callbacks::on\_readed\_arg

Additional argument for the read callback.

Definition at line 204 of file low\_saurion.h.

#### 6.5.2.9 on\_wrote

void(\* saurion::saurion\_callbacks::on\_wrote) (const int fd, void \*arg)

Callback for handling write events.

## **Parameters**

| f | d   | File descriptor of the socket.     |
|---|-----|------------------------------------|
| â | arg | Additional user-provided argument. |

Definition at line 212 of file low\_saurion.h.

#### 6.5.2.10 on\_wrote\_arg

void\* saurion::saurion\_callbacks::on\_wrote\_arg

Additional argument for the write callback.

Definition at line 213 of file low\_saurion.h.

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

• /\_\_w/saurion/saurion/include/low\_saurion.h

## 6.6 saurion callbacks Struct Reference

Structure containing callback functions to handle socket events.

```
#include <low_saurion.h>
```

#### **Public Attributes**

void(\* on connected )(const int fd, void \*arg)

Callback for handling new connections.

- void \* on\_connected\_arg
- void(\* on\_readed )(const int fd, const void \*const content, const ssize\_t len, void \*arg)

Callback for handling read events.

- void \* on\_readed\_arg
- void(\* on\_wrote )(const int fd, void \*arg)

Callback for handling write events.

- void \* on wrote arg
- void(\* on\_closed )(const int fd, void \*arg)

Callback for handling socket closures.

- void \* on\_closed\_arg
- void(\* on\_error )(const int fd, const char \*const content, const ssize\_t len, void \*arg)

Callback for handling error events.

void \* on\_error\_arg

#### 6.6.1 Detailed Description

Structure containing callback functions to handle socket events.

This structure holds pointers to callback functions for handling events such as connection establishment, reading, writing, closing, and errors on sockets. Each callback has an associated argument pointer that can be passed along when the callback is invoked.

Definition at line 31 of file low saurion.h.

## 6.6.2 Member Data Documentation

#### 6.6.2.1 on\_closed

```
void(* saurion_callbacks::on_closed) (const int fd, void *arg)
```

Callback for handling socket closures.

#### **Parameters**

| fd  | File descriptor of the closed socket. |
|-----|---------------------------------------|
| arg | Additional user-provided argument.    |

Definition at line 71 of file low\_saurion.h.

#### 6.6.2.2 on\_closed\_arg

```
void* saurion_callbacks::on_closed_arg
```

Additional argument for the close callback.

Definition at line 73 of file low\_saurion.h.

#### 6.6.2.3 on\_connected

```
void(* saurion_callbacks::on_connected) (const int fd, void *arg)
```

Callback for handling new connections.

#### **Parameters**

| fd  | File descriptor of the connected socket. |
|-----|--|
| arg | Additional user-provided argument.       |

Definition at line 39 of file low\_saurion.h.

## 6.6.2.4 on\_connected\_arg

```
void* saurion_callbacks::on_connected_arg
```

Additional argument for the connection callback.

Definition at line 41 of file low\_saurion.h.

#### 6.6.2.5 on\_error

```
void(* saurion\_callbacks::on\_error) (const int fd, const char *const content, const ssize_\leftarrow t len, void *arg)
```

Callback for handling error events.

#### **Parameters**

| fd                  | File descriptor of the socket where the error occurred. |
|---------------------|---|
|                     | Pointer to the error message.                           |
| Generated by<br>IEN | Doxygen<br>Length of the error message.                 |
| arg                 | Additional user-provided argument.                      |

Definition at line 83 of file low\_saurion.h.

#### 6.6.2.6 on\_error\_arg

```
void* saurion_callbacks::on_error_arg
```

Additional argument for the error callback.

Definition at line 86 of file low\_saurion.h.

#### 6.6.2.7 on\_readed

 $\label{local_const_con$ 

Callback for handling read events.

#### **Parameters**

| fd      | File descriptor of the socket.     |
|---------|------------------------------------|
| content | Pointer to the data that was read. |
| len     | Length of the data that was read.  |
| arg     | Additional user-provided argument. |

Definition at line 51 of file low\_saurion.h.

#### 6.6.2.8 on readed arg

```
void* saurion_callbacks::on_readed_arg
```

Additional argument for the read callback.

Definition at line 54 of file low\_saurion.h.

#### 6.6.2.9 on\_wrote

```
void(* saurion_callbacks::on_wrote) (const int fd, void *arg)
```

Callback for handling write events.

#### **Parameters**

| fd  | File descriptor of the socket.     |
|-----|------------------------------------|
| arg | Additional user-provided argument. |

Definition at line 62 of file low\_saurion.h.

#### 6.6.2.10 on\_wrote\_arg

```
void* saurion_callbacks::on_wrote_arg
```

Additional argument for the write callback.

Definition at line 63 of file low\_saurion.h.

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

• /\_\_w/saurion/saurion/include/low\_saurion.h

# 6.7 saurion\_wrapper Struct Reference

Collaboration diagram for saurion\_wrapper:

#### **Public Attributes**

- struct saurion \* s
- uint32\_t sel

## 6.7.1 Detailed Description

Definition at line 51 of file low\_saurion.c.

#### 6.7.2 Member Data Documentation

### 6.7.2.1 s

```
struct saurion* saurion_wrapper::s
```

Definition at line 53 of file low\_saurion.c.

## 6.7.2.2 sel

```
uint32_t saurion_wrapper::sel
```

Definition at line 54 of file low\_saurion.c.

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

• /\_\_w/saurion/saurion/src/low\_saurion.c

## 6.8 task Struct Reference

Collaboration diagram for task:

## **Public Attributes**

- void(\* function )(void \*)
- void \* argument
- struct task \* next

## 6.8.1 Detailed Description

Definition at line 11 of file threadpool.c.

## 6.8.2 Member Data Documentation

## 6.8.2.1 argument

```
void* task::argument
```

Definition at line 14 of file threadpool.c.

#### 6.8.2.2 function

```
void(* task::function) (void *)
```

Definition at line 13 of file threadpool.c.

#### 6.8.2.3 next

```
struct task* task::next
```

Definition at line 15 of file threadpool.c.

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

• /\_\_w/saurion/saurion/src/threadpool.c

# 6.9 threadpool Struct Reference

Collaboration diagram for threadpool:

## **Public Attributes**

- pthread\_t \* threads
- size\_t num\_threads
- struct task \* task\_queue\_head
- struct task \* task\_queue\_tail
- pthread\_mutex\_t queue\_lock
- pthread\_cond\_t queue\_cond
- pthread\_cond\_t empty\_cond
- int stop
- int started

## 6.9.1 Detailed Description

Definition at line 18 of file threadpool.c.

### 6.9.2 Member Data Documentation

## 6.9.2.1 empty\_cond

```
pthread_cond_t threadpool::empty_cond
```

Definition at line 26 of file threadpool.c.

#### 6.9.2.2 num\_threads

```
size_t threadpool::num_threads
```

Definition at line 21 of file threadpool.c.

#### 6.9.2.3 queue\_cond

```
\verb|pthread_cond_t| threadpool::queue\_cond|
```

Definition at line 25 of file threadpool.c.

#### 6.9.2.4 queue\_lock

```
pthread_mutex_t threadpool::queue_lock
```

Definition at line 24 of file threadpool.c.

#### 6.9.2.5 started

int threadpool::started

Definition at line 28 of file threadpool.c.

## 6.9.2.6 stop

```
int threadpool::stop
```

Definition at line 27 of file threadpool.c.

#### 6.9.2.7 task\_queue\_head

```
struct task* threadpool::task_queue_head
```

Definition at line 22 of file threadpool.c.

## 6.9.2.8 task\_queue\_tail

```
struct task* threadpool::task_queue_tail
```

Definition at line 23 of file threadpool.c.

#### 6.9.2.9 threads

```
pthread_t* threadpool::threads
```

Definition at line 20 of file threadpool.c.

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

• /\_\_w/saurion/saurion/src/threadpool.c

# **Chapter 7**

# **File Documentation**

# 7.1 /\_w/saurion/saurion/include/linked\_list.h File Reference

```
#include <stddef.h>
```

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

#### **Functions**

- int list\_insert (struct Node \*\*head, void \*ptr, size\_t amount, void \*\*children)
- void list\_delete\_node (struct Node \*\*head, const void \*const ptr)
- void list\_free (struct Node \*\*head)

#### 7.1.1 Function Documentation

## 7.1.1.1 list\_delete\_node()

#### Definition at line 106 of file linked\_list.c.

```
00107 {
00108
       pthread_mutex_lock (&list_mutex);
       struct Node *current = *head;
struct Node *prev = NULL;
00109
00110
00111
00112
        if (current && current->ptr == ptr)
00113
            *head = current->next;
00114
00115
            free_node (current);
            pthread_mutex_unlock (&list_mutex);
00116
00117
            return;
00118
00119
00120
       while (current && current->ptr != ptr)
00121
00122
            prev = current;
00123
            current = current->next;
00124
```

54 File Documentation

```
00125
00126
       if (!current)
00127
           pthread_mutex_unlock (&list_mutex);
00128
00129
            return;
00130
00131
00132
       prev->next = current->next;
00133
       free_node (current);
       pthread_mutex_unlock (&list_mutex);
00134
00135 }
```

#### 7.1.1.2 list\_free()

### Definition at line 138 of file linked\_list.c.

```
00139 {
00140
         pthread_mutex_lock (&list_mutex);
        struct Node *current = *head;
struct Node *next;
00141
00142
00143
        while (current)
00145
         {
00146
           next = current->next;
            free_node (current);
current = next;
00147
00148
00149
00150
00151
        *head = NULL;
00152
        pthread_mutex_unlock (&list_mutex);
00153 }
```

## 7.1.1.3 list\_insert()

#### Definition at line 65 of file linked\_list.c.

```
00066 {
00067
       struct Node *new_node = create_node (ptr, amount, children);
00068
       if (!new_node)
        {
00069
00070
           return 1:
00071
00072
       pthread_mutex_lock (&list_mutex);
00073
        if (!*head)
00074
        {
           *head = new_node;
00075
00076
           pthread_mutex_unlock (&list_mutex);
          return 0;
00077
00078
00079
       struct Node *temp = *head;
08000
       while (temp->next)
00081
00082
           temp = temp->next;
00083
00084
       temp->next = new_node;
00085
       pthread_mutex_unlock (&list_mutex);
00086
       return 0;
00087 }
```

7.2 linked list.h 55

# 7.2 linked\_list.h

#### Go to the documentation of this file.

```
00001 #ifndef LINKED_LIST_H
00002 #define LINKED_LIST_H
00003
00004 #ifdef __cplusplus
00005 extern "C"
00006 {
00007 #endif
80000
00009 #include <stddef.h>
00010
00011
        struct Node;
00012
00013
        int list_insert (struct Node **head, void *ptr, size_t amount,
00014
                            void **children);
00015
00016
        void list delete node (struct Node **head, const void *const ptr);
00018
        void list_free (struct Node **head);
00019
00020 #ifdef __cplusplus
00021 }
00022 #endif
00023
00024 #endif // !LINKED_LIST_H
```

# 7.3 /\_w/saurion/saurion/include/low\_saurion.h File Reference

```
#include <pthread.h>
#include <stdint.h>
#include <sys/types.h>
```

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

## Classes

· struct saurion

Main structure for managing io\_uring and socket events.

struct saurion::saurion\_callbacks

Structure containing callback functions to handle socket events.

• struct saurion callbacks

Structure containing callback functions to handle socket events.

### **Macros**

- #define \_POSIX\_C\_SOURCE 200809L
- #define PACKING\_SZ 32

Defines the memory alignment size for structures in the saurion class.

## **Functions**

- int EXTERNAL\_set\_socket (int p)
- struct saurion \* saurion\_create (uint32\_t n\_threads)

Creates an instance of the saurion structure.

int saurion\_start (struct saurion \*s)

Starts event processing in the saurion structure.

void saurion\_stop (const struct saurion \*s)

Stops event processing in the saurion structure.

void saurion\_destroy (struct saurion \*s)

Destroys the saurion structure and frees all associated resources.

void saurion\_send (struct saurion \*s, const int fd, const char \*const msg)

Sends a message through a socket using io\_uring.

56 File Documentation

#### **Variables**

```
    void(* on_connected )(const int fd, void *arg)
```

Callback for handling new connections.

- void \* on\_connected\_arg
- void(\* on\_readed )(const int fd, const void \*const content, const ssize\_t len, void \*arg)

Callback for handling read events.

- void \* on\_readed\_arg
- void(\* on\_wrote )(const int fd, void \*arg)

Callback for handling write events.

- void \* on\_wrote\_arg
- void(\* on\_closed )(const int fd, void \*arg)

Callback for handling socket closures.

- void \* on\_closed\_arg
- void(\* on\_error )(const int fd, const char \*const content, const ssize\_t len, void \*arg)

Callback for handling error events.

- void \* on\_error\_arg
- struct io\_uring \* rings
- pthread\_mutex\_t \* m\_rings
- int ss
- int \* efds
- struct Node \* list
- pthread\_mutex\_t status\_m
- pthread\_cond\_t status\_c
- · int status
- struct threadpool \* pool
- uint32\_t n\_threads
- uint32\_t next

## 7.3.1 Variable Documentation

#### 7.3.1.1 efds

int\* efds

Eventfd descriptors used for internal signaling between threads.

Definition at line 7 of file low\_saurion.h.

## 7.3.1.2 list

struct Node\* list

Linked list for storing active requests.

Definition at line 9 of file low\_saurion.h.

## 7.3.1.3 m\_rings

```
{\tt pthread\_mutex\_t*} \ {\tt m\_rings}
```

Array of mutexes to protect the io\_uring rings.

Definition at line 3 of file low\_saurion.h.

## 7.3.1.4 n\_threads

```
uint32_t n_threads
```

Number of threads in the thread pool.

Definition at line 19 of file low\_saurion.h.

#### 7.3.1.5 next

```
uint32_t next
```

Index of the next io\_uring ring to which an event will be added.

Definition at line 21 of file low\_saurion.h.

## 7.3.1.6 on\_closed

Callback for handling socket closures.

#### **Parameters**

| fd  | File descriptor of the closed socket. |
|-----|---------------------------------------|
| arg | Additional user-provided argument.    |

Definition at line 38 of file low\_saurion.h.

## 7.3.1.7 on\_closed\_arg

```
void * on_closed_arg
```

58 File Documentation

Additional argument for the close callback.

Definition at line 40 of file low\_saurion.h.

#### 7.3.1.8 on\_connected

Callback for handling new connections.

#### **Parameters**

| fd  | File descriptor of the connected socket. |
|-----|--|
| arg | Additional user-provided argument.       |

Definition at line 6 of file low\_saurion.h.

## 7.3.1.9 on\_connected\_arg

```
void * on_connected_arg
```

Additional argument for the connection callback.

Definition at line 8 of file low\_saurion.h.

## 7.3.1.10 on\_error

Callback for handling error events.

#### **Parameters**

| fd      | File descriptor of the socket where the error occurred. |
|---------|---|
| content | Pointer to the error message.                           |
| len     | Length of the error message.                            |
| arg     | Additional user-provided argument.                      |

Definition at line 50 of file low\_saurion.h.

#### 7.3.1.11 on\_error\_arg

```
void * on_error_arg
```

Additional argument for the error callback.

Definition at line 53 of file low\_saurion.h.

#### 7.3.1.12 on\_readed

Callback for handling read events.

#### **Parameters**

| fd      | File descriptor of the socket.     |
|---------|------------------------------------|
| content | Pointer to the data that was read. |
| len     | Length of the data that was read.  |
| arg     | Additional user-provided argument. |

Definition at line 18 of file low\_saurion.h.

#### 7.3.1.13 on\_readed\_arg

```
void * on_readed_arg
```

Additional argument for the read callback.

Definition at line 21 of file low\_saurion.h.

#### 7.3.1.14 on\_wrote

Callback for handling write events.

60 File Documentation

#### **Parameters**

| fd  | File descriptor of the socket.     |
|-----|------------------------------------|
| arg | Additional user-provided argument. |

Definition at line 29 of file low\_saurion.h.

## 7.3.1.15 on\_wrote\_arg

```
void * on_wrote_arg
```

Additional argument for the write callback.

Definition at line 30 of file low\_saurion.h.

## 7.3.1.16 pool

```
struct threadpool* pool
```

Thread pool for executing tasks in parallel.

Definition at line 17 of file low\_saurion.h.

## 7.3.1.17 rings

```
struct io_uring* rings
```

Array of io\_uring structures for managing the event queue.

Definition at line 1 of file low\_saurion.h.

## 7.3.1.18 ss

int ss

Server socket descriptor for accepting connections.

Definition at line 5 of file low\_saurion.h.

7.4 low\_saurion.h

#### 7.3.1.19 status

```
int status
```

Current status of the structure (e.g., running, stopped).

Definition at line 15 of file low\_saurion.h.

#### 7.3.1.20 status\_c

```
pthread_cond_t status_c
```

Condition variable to signal changes in the structure's state.

Definition at line 13 of file low saurion.h.

#### 7.3.1.21 status\_m

```
pthread_mutex_t status_m
```

Mutex to protect the state of the structure.

Definition at line 11 of file low\_saurion.h.

## 7.4 low\_saurion.h

```
00104 #ifndef LOW_SAURION_H
00105 #define LOW_SAURION_H
00106
00107 #define _POSIX_C_SOURCE 200809L
00109 #include <pthread.h> // for pthread_mutex_t, pthread_cond_t
00110 #include <stdint.h> // for uint32_t
00111 #include <sys/types.h> // for ssize_t
00112
00113 #ifdef __cplusplus
00114 extern "C"
00115 {
00116 #endif
00117
00139 #define PACKING_SZ 32
00140
00148
         struct saurion
00149
           struct io_uring *rings;
00153
         pthread_mutex_t *m_rings;
00155
            int ss;
00157
           int *efds:
00159
           struct Node *list;
00161
            pthread_mutex_t status_m;
00163
            pthread_cond_t status_c;
00165
            int status;
00167
            struct threadpool *pool;
00169
00171
            uint32_t n_threads;
uint32_t next;
00172
00181
            struct saurion_callbacks
```

62 File Documentation

```
void (*on_connected) (const int fd, void *arg);
00189
00191
            void *on_connected_arg;
00192
            void (*on_readed) (const int fd, const void *const content,
00202
                                const ssize_t len, void *arg);
            void *on_readed_arg;
00205
00212
            void (*on_wrote) (const int fd, void *arg);
00213
            void *on_wrote_arg;
00221
            void (*on_closed) (const int fd, void *arg);
00223
            void *on_closed_arg;
00224
            void (*on_error) (const int fd, const char *const content,
00234
                               const ssize_t len, void *arg);
00236
            void *on_error_arg;
       } __attribute__ ((aligned (PACKING_SZ))) cb;
} __attribute__ ((aligned (PACKING_SZ)));
00237
00238
00239
00243
        int EXTERNAL_set_socket (int p);
00244
00257
        [[nodiscard]]
00258
       struct saurion *saurion_create (uint32_t n_threads);
00259
00272
        [[nodiscard]]
00273
        int saurion_start (struct saurion *s);
00274
00285
       void saurion_stop (const struct saurion *s);
00286
00299
       void saurion_destroy (struct saurion *s);
00300
00313
       void saurion_send (struct saurion *s, const int fd, const char *const msg);
00314
00315 #ifdef __cplusplus
00316
00317 #endif
00318
00319 #endif // !LOW_SAURION_H
```

# 7.5 / w/saurion/saurion/include/low saurion secret.h File Reference

```
#include <bits/types/struct_iovec.h>
#include <stddef.h>
#include <stdint.h>
Include dependency graph for low saurion secret.h:
```

#### **Functions**

- int allocate\_iovec (struct iovec \*iov, size\_t amount, size\_t pos, size\_t size, void \*\*chd\_ptr)
- int initialize\_iovec (struct iovec \*iov, size\_t amount, size\_t pos, const void \*msg, size\_t size, uint8\_t h)

Initializes a specified iovec structure with a message fragment.

int set\_request (struct request \*\*r, struct Node \*\*I, size\_t s, const void \*m, uint8\_t h)

Sets up a request and allocates iovec structures for data handling in liburing.

int read\_chunk (void \*\*dest, size\_t \*len, struct request \*const req)

Reads a message chunk from the request's iovec buffers, handling messages that may span multiple iovec entries.

void free\_request (struct request \*req, void \*\*children\_ptr, size\_t amount)

## 7.6 low saurion secret.h

```
00001 #ifndef LOW_SAURION_SECRET_H
00002 #define LOW_SAURION_SECRET_H
00003
00004 #include <bits/types/struct_iovec.h>
```

```
00005 #include <stddef.h>
00006 #include <stdint.h>
00007
00008 #ifdef __cplusplus
00009 extern "C" {
00010 #endif
00015 #pragma GCC diagnostic push
00016 #pragma GCC diagnostic ignored "-Wpedantic"
00017 struct request {
00018 void *prev;
00019
       size_t prev_size;
00020 size_t prev_remain;
00021
00022
       size_t next_iov;
       size_t next_offset;
00023
       int event_type;
00024 size_t iovec_count;
00025 int client socket;
00026
       struct iovec iov[];
00028 #pragma GCC diagnostic pop
00062 [[nodiscard]]
00063 int allocate_iovec(struct iovec *iov, size_t amount, size_t pos, size_t size, void **chd_ptr);
00064
00097 [[nodiscard]]
00098 int initialize_iovec(struct iovec *iov, size_t amount, size_t pos, const void *msq, size_t size,
                           uint8_t h);
00100
00127 [[nodiscard]]
00128 int set_request (struct request **r, struct Node **1, size_t s, const void *m, uint8_t h);
00129
00165 [[nodiscard]]
00166 int read_chunk(void **dest, size_t *len, struct request *const req);
00168 void free_request (struct request *req, void **children_ptr, size_t amount);
00172 #ifdef __cplusplus
00173 }
00174 #endif
00175
00176 #endif // !LOW_SAURION_SECRET_H
```

# 7.7 /\_w/saurion/saurion/include/saurion.hpp File Reference

```
#include <stdint.h>
#include <sys/types.h>
```

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

#### **Classes**

• class Saurion

# 7.8 saurion.hpp

```
00001 #ifndef SAURION_HPP
00002 #define SAURION_HPP
00003
00004 #include <stdint.h>
                            // for uint32_t
00005 #include <sys/types.h> // for ssize_t
00006
00007 class Saurion
00008 {
00009 public:
       using ConnectedCb = void (*) (const int, void *);
       using ReadedCb
00011
00012
            = void (*) (const int, const void *const, const ssize_t, void *);
00013
       using WroteCb = void (*) (const int, void *);
00014
       using ClosedCb = void (*) (const int, void *);
00015
       using ErrorCb
00016
           = void (*) (const int, const char *const, const ssize_t, void *);
```

64 File Documentation

```
explicit Saurion (const uint32_t thds, const int sck) noexcept;
00019
00020
00021
        Saurion (const Saurion &) = delete;
00022
        Saurion (Saurion &&) = delete;
        Saurion & operator = (const Saurion &) = delete;
Saurion & operator = (Saurion & &) = delete;
00023
00024
00025
00026
        void init () noexcept;
00027
        void stop () const noexcept;
00028
00029
        Saurion *on_connected (ConnectedCb ncb, void *arg) noexcept;
00030
        Saurion *on_readed (ReadedCb ncb, void *arg) noexcept;
00031
        Saurion *on_wrote (WroteCb ncb, void *arg) noexcept;
00032
        Saurion *on_closed (ClosedCb ncb, void *arg) noexcept;
00033
        Saurion *on_error (ErrorCb ncb, void *arg) noexcept;
00034
00035
        void send (const int fd, const char *const msg) noexcept;
00036
00037 private:
00038
       struct saurion *s;
00039 };
00040
00041 #endif // !SAURTON HPP
```

# 7.9 /\_w/saurion/saurion/include/threadpool.h File Reference

#include <stddef.h>

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

#### **Functions**

- struct threadpool \* threadpool\_create (size\_t num\_threads)
- struct threadpool \* threadpool\_create\_default (void)
- void threadpool\_init (struct threadpool \*pool)
- void threadpool\_add (struct threadpool \*pool, void(\*function)(void \*), void \*argument)
- void threadpool\_stop (struct threadpool \*pool)
- int threadpool\_empty (struct threadpool \*pool)
- void threadpool wait empty (struct threadpool \*pool)
- void threadpool destroy (struct threadpool \*pool)

# 7.10 threadpool.h

```
00006 #ifndef THREADPOOL_H
00007 #define THREADPOOL_H
00008
00009 #include <stddef.h> // for size_t
00010
#ifdef __cplusplus
00012 extern "C"
00013 {
00014 #endif
00015
00016
       struct threadpool;
00017
00018
       struct threadpool *threadpool_create (size_t num_threads);
00020
       struct threadpool *threadpool_create_default (void);
00021
00022
       void threadpool_init (struct threadpool *pool);
00023
00024
       void threadpool_add (struct threadpool *pool, void (*function) (void *),
00025
                              void *argument);
```

```
00027
       void threadpool_stop (struct threadpool *pool);
00028
00029
       int threadpool_empty (struct threadpool *pool);
00030
00031
       void threadpool_wait_empty (struct threadpool *pool);
00032
       void threadpool_destroy (struct threadpool *pool);
00034
00035 #ifdef __cplusplus
00036 }
00037 #endif
00038
00039 #endif // !THREADPOOL_H
00040
```

# 7.11 / w/saurion/saurion/src/linked list.c File Reference

```
#include "linked_list.h"
#include <pthread.h>
#include <stdlib.h>
Include dependency graph for linked_list.c:
```

#### **Classes**

• struct Node

#### **Functions**

- struct Node \* create\_node (void \*ptr, size\_t amount, void \*\*children)
- int list\_insert (struct Node \*\*head, void \*ptr, size\_t amount, void \*\*children)
- void free\_node (struct Node \*current)
- void list\_delete\_node (struct Node \*\*head, const void \*const ptr)
- void list\_free (struct Node \*\*head)

### **Variables**

• pthread\_mutex\_t list\_mutex = PTHREAD\_MUTEX\_INITIALIZER

#### 7.11.1 Function Documentation

#### 7.11.1.1 create\_node()

```
struct Node * create_node (
               void * ptr,
               size_t amount,
               void ** children )
Definition at line 17 of file linked list.c.
00018 {
00019
        struct Node *new_node = (struct Node *)malloc (sizeof (struct Node));
00020
        if (!new_node)
00021
         {
00022
            return NULL;
00023
        new_node->ptr = ptr;
new_node->size = amount;
00024
00025
        new_node->children = NULL;
00026
00027
        if (amount <= 0)</pre>
00028
         {
00029
            new_node->next = NULL;
00030
            return new_node;
00031
00032
        new_node->children
00033
             = (struct Node **) malloc (sizeof (struct Node *) * amount);
00034
        if (!new_node->children)
00035
00036
            free (new_node);
00037
            return NULL;
00038
        for (size_t i = 0; i < amount; ++i)</pre>
00039
00040
00041
            new_node->children[i] = (struct Node *)malloc (sizeof (struct Node));
00042
00043
             if (!new_node->children[i])
00044
00045
                 for (size_t j = 0; j < i; ++j)
00047
                    free (new_node->children[j]);
00048
00049
                free (new_node);
00050
                return NULL;
00051
00052
00053
        for (size_t i = 0; i < amount; ++i)</pre>
00054
00055
            new node->children[i]->size = 0;
            new_node->children[i]->next = NULL;
00056
            new_node->children[i]->ptr = children[i];
00057
            new_node->children[i]->children = NULL;
00058
00059
       new_node->next = NULL;
00060
00061
        return new_node;
00062 }
```

#### 7.11.1.2 free\_node()

#### Definition at line 90 of file linked list.c.

```
00091 {
        if (current->size > 0)
00093
         {
00094
            for (size_t i = 0; i < current->size; ++i)
00095
                free (current->children[i]->ptr);
00096
00097
                free (current->children[i]);
00098
00099
            free (current->children);
00100
00101
       free (current->ptr);
00102
       free (current);
00103 }
```

#### 7.11.1.3 list\_delete\_node()

```
void list_delete_node (
             struct Node ** head,
              const void *const ptr )
Definition at line 106 of file linked_list.c.
00107
00108
       pthread mutex lock (&list mutex);
       struct Node *current = *head;
00109
00110
       struct Node *prev = NULL;
00111
00112
       if (current && current->ptr == ptr)
00113
           *head = current->next;
00114
00115
           free node (current);
00116
           pthread_mutex_unlock (&list_mutex);
00117
           return;
00118
00119
00120
       while (current && current->ptr != ptr)
00121
        {
00122
           prev = current;
00123
           current = current->next;
       }
00124
00125
00126
       if (!current)
00127
       {
00128
         pthread_mutex_unlock (&list_mutex);
00129
           return;
00130
00131
00132 prev->next = current->next;
00133
       free_node (current);
00134
       pthread_mutex_unlock (&list_mutex);
00135 }
```

#### 7.11.1.4 list\_free()

#### Definition at line 138 of file linked\_list.c.

```
00139 {
00140
         pthread_mutex_lock (&list_mutex);
        struct Node *current = *head;
struct Node *next;
00141
00142
00143
00144
        while (current)
00145
00146
             next = current->next;
             free_node (current);
current = next;
00147
00148
00149
00150
00151
        *head = NULL;
00152 pthread_mutex_unlock (&list_mutex);
00153 }
```

#### 7.11.1.5 list\_insert()

```
size_t amount,
void ** children )
```

#### Definition at line 65 of file linked\_list.c.

```
00066 {
        struct Node *new_node = create_node (ptr, amount, children);
00067
00068
        if (!new_node)
00069
         {
00070
            return 1;
00071
00072
        pthread_mutex_lock (&list_mutex);
00073
        if (!*head)
00074
        {
00075
           *head = new_node;
00076
           pthread_mutex_unlock (&list_mutex);
00077
           return 0;
00078
00079
       struct Node *temp = *head;
08000
       while (temp->next)
00081
        {
00082
           temp = temp->next;
00083
00084
       temp->next = new_node;
00085
       pthread_mutex_unlock (&list_mutex);
00086
        return 0;
00087 }
```

#### 7.11.2 Variable Documentation

#### 7.11.2.1 list\_mutex

```
pthread_mutex_t list_mutex = PTHREAD_MUTEX_INITIALIZER
```

Definition at line 14 of file linked\_list.c.

# 7.12 linked list.c

#### Go to the documentation of this file.

```
00001 #include "linked_list.h"
00002
00003 #include <pthread.h>
00004 #include <stdlib.h>
00005
00006 struct Node
00007 {
       void *ptr;
00008
00009
       size_t size;
00010
       struct Node **children;
00011
       struct Node *next;
00012 };
00013
00014 pthread_mutex_t list_mutex = PTHREAD_MUTEX_INITIALIZER;
00015
00016 struct Node *
00017 create_node (void *ptr, size_t amount, void **children)
00018 {
       struct Node *new_node = (struct Node *)malloc (sizeof (struct Node));
00019
00020
       if (!new_node)
00021
00022
           return NULL;
00023
00024
       new_node->ptr = ptr;
       new_node->size = amount;
00025
00026
        new_node->children = NULL;
00027
       if (amount <= 0)</pre>
00028
00029
            new_node->next = NULL;
```

7.12 linked\_list.c 69

```
00030
            return new_node;
00031
00032
        new_node->children
            = (struct Node **) malloc (sizeof (struct Node *) * amount);
00033
00034
        if (!new_node->children)
00035
         {
            free (new_node);
00037
            return NULL;
00038
        for (size_t i = 0; i < amount; ++i)</pre>
00039
00040
00041
            new node->children[i] = (struct Node *)malloc (sizeof (struct Node));
00042
00043
            if (!new_node->children[i])
00044
00045
                for (size_t j = 0; j < i; ++j)
00046
00047
                    free (new_node->children[j]);
00048
00049
                free (new_node);
00050
                return NULL;
00051
00052
        for (size_t i = 0; i < amount; ++i)</pre>
00053
00054
00055
            new_node->children[i]->size = 0;
00056
            new_node->children[i]->next = NULL;
00057
            new_node->children[i]->ptr = children[i];
00058
            new_node->children[i]->children = NULL;
00059
00060
       new node->next = NULL:
00061
        return new_node;
00062 }
00063
00064 int.
00065 list_insert (struct Node **head, void *ptr, size_t amount, void **children)
00066 {
        struct Node *new_node = create_node (ptr, amount, children);
00068
        if (!new_node)
00069
00070
            return 1;
00071
00072
        pthread_mutex_lock (&list_mutex);
00073
        if (!*head)
00074
00075
            *head = new_node;
00076
         return 0;
}
          pthread_mutex_unlock (&list_mutex);
00077
00078
00079
        struct Node *temp = *head;
        while (temp->next)
08000
00081
00082
           temp = temp->next;
00083
       temp->next = new_node;
00084
00085
        pthread_mutex_unlock (&list_mutex);
00086
        return 0;
00087 }
00088
00089 void
00090 free node (struct Node *current)
00091 {
00092
        if (current->size > 0)
00093
00094
            for (size_t i = 0; i < current->size; ++i)
00095
             {
00096
                free (current->children[i]->ptr);
00097
                free (current->children[i]);
00098
00099
           free (current->children);
00100
00101
       free (current->ptr);
00102
       free (current);
00103 }
00104
00105 void
00106 list_delete_node (struct Node **head, const void *const ptr)
00107 {
00108
        pthread_mutex_lock (&list_mutex);
        struct Node *current = *head;
struct Node *prev = NULL;
00109
00110
00111
00112
        if (current && current->ptr == ptr)
00113
00114
            *head = current->next;
00115
            free node (current);
00116
            pthread_mutex_unlock (&list_mutex);
```

```
return;
00118
00119
00120
        while (current && current->ptr != ptr)
00121
00122
            prev = current;
00123
            current = current->next;
         }
00124
00125
00126
        if (!current)
        {
00127
         pthread_mutex_unlock (&list_mutex);
00128
00129
            return:
00130
00131
00132 prev->next = current->next;
00133 free node (current);
00134
       pthread_mutex_unlock (&list_mutex);
00135 }
00136
00137 void
00138 list_free (struct Node **head)
00139 {
        pthread_mutex_lock (&list_mutex);
00140
        struct Node *current = *head;
struct Node *next;
00141
00143
00144
        while (current)
00145
00146
            next = current->next;
00147
            free node (current);
00148
            current = next;
00149
00150
00151
        *head = NULL;
       pthread_mutex_unlock (&list_mutex);
00152
00153 }
```

# 7.13 / w/saurion/saurion/src/low saurion.c File Reference

```
#include "low_saurion.h"
#include "config.h"
#include "linked_list.h"
#include "threadpool.h"
#include <arpa/inet.h>
#include <bits/socket-constants.h>
#include <liburing.h>
#include <liburing/io_uring.h>
#include <nanologger.h>
#include <netinet/in.h>
#include <pthread.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/eventfd.h>
#include <sys/socket.h>
#include <sys/uio.h>
#include <time.h>
#include <unistd.h>
Include dependency graph for low_saurion.c:
```

#### **Classes**

- · struct request
- · struct saurion\_wrapper

#### **Macros**

- #define EV\_ACC 0
- #define EV REA 1
- #define EV WRI 2
- #define EV WAI 3
- #define EV ERR 4
- #define MIN(a, b) ((a) < (b) ? (a) : (b))
- #define MAX(a, b) ((a) > (b) ? (a) : (b))

#### **Functions**

- static uint32 t next (struct saurion \*s)
- static uint64 t htonll (uint64 t value)
- static uint64 t ntohll (uint64 t value)
- void free\_request (struct request \*req, void \*\*children\_ptr, size\_t amount)
- int initialize\_iovec (struct iovec \*iov, size\_t amount, size\_t pos, const void \*msg, size\_t size, uint8\_t h)

Initializes a specified iovec structure with a message fragment.

- int allocate\_iovec (struct iovec \*iov, size\_t amount, size\_t pos, size\_t size, void \*\*chd\_ptr)
- int set\_request (struct request \*\*r, struct Node \*\*I, size\_t s, const void \*m, uint8\_t h)

Sets up a request and allocates iovec structures for data handling in liburing.

- static void add\_accept (struct saurion \*const s, struct sockaddr\_in \*const ca, socklen\_t \*const cal)
- static void add\_efd (struct saurion \*const s, const int client\_socket, int sel)
- static void add read (struct saurion \*const s, const int client socket)
- static void add\_read\_continue (struct saurion \*const s, struct request \*oreq, const int sel)
- static void add\_write (struct saurion \*const s, int fd, const char \*const str, const int sel)
- static void handle\_accept (const struct saurion \*const s, const int fd)
- int read\_chunk (void \*\*dest, size\_t \*len, struct request \*const req)

Reads a message chunk from the request's iovec buffers, handling messages that may span multiple iovec entries.

- static void handle\_read (struct saurion \*const s, struct request \*const req)
- static void <a href="mailto:handle\_write">handle\_write</a> (const struct saurion \*const s, const int fd)
- static void handle\_error (const struct saurion \*const s, const struct request \*const req)
- static void handle\_close (const struct saurion \*const s, const struct request \*const req)
- int EXTERNAL\_set\_socket (const int p)
- struct saurion \* saurion\_create (uint32\_t n\_threads)

Creates an instance of the saurion structure.

- static int saurion\_worker\_master\_loop\_it (struct saurion \*const s, struct sockaddr\_in \*client\_addr, socklen\_t \*client\_addr\_len)
- void saurion\_worker\_master (void \*arg)
- static int saurion\_worker\_slave\_loop\_it (struct saurion \*const s, const int sel)
- void saurion\_worker\_slave (void \*arg)
- int saurion\_start (struct saurion \*const s)

Starts event processing in the saurion structure.

void saurion\_stop (const struct saurion \*const s)

Stops event processing in the saurion structure.

void saurion\_destroy (struct saurion \*const s)

Destroys the saurion structure and frees all associated resources.

void saurion\_send (struct saurion \*const s, const int fd, const char \*const msg)

Sends a message through a socket using io\_uring.

### **Variables**

- pthread\_mutex\_t print\_mutex
- struct timespec TIMEOUT\_RETRY\_SPEC = { 0, TIMEOUT\_RETRY \* 1000L }

# 7.13.1 Macro Definition Documentation

# 7.13.1.1 EV\_ACC

```
#define EV_ACC 0
```

Definition at line 26 of file low\_saurion.c.

# 7.13.1.2 EV\_ERR

```
#define EV_ERR 4
```

Definition at line 30 of file low\_saurion.c.

#### 7.13.1.3 EV\_REA

```
#define EV_REA 1
```

Definition at line 27 of file low\_saurion.c.

### 7.13.1.4 EV\_WAI

```
#define EV_WAI 3
```

Definition at line 29 of file low\_saurion.c.

# 7.13.1.5 EV\_WRI

#define EV\_WRI 2

Definition at line 28 of file low\_saurion.c.

#### 7.13.1.6 MAX

Definition at line 46 of file low saurion.c.

#### 7.13.1.7 MIN

Definition at line 45 of file low saurion.c.

#### 7.13.2 Function Documentation

#### 7.13.2.1 add\_accept()

# Definition at line 250 of file low\_saurion.c.

```
00253
        int res = ERROR_CODE;
00254
        pthread_mutex_lock (&s->m_rings[0]);
00255
        while (res != SUCCESS_CODE)
00256
          {
00257
            struct io_uring_sqe *sqe = io_uring_get_sqe (&s->rings[0]);
00258
            while (!sqe)
00259
              {
00260
                 sqe = io_uring_get_sqe (&s->rings[0]);
00261
                 nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00262
            struct request *req = NULL;
00263
00264
            if (!set_request (&req, &s->list, 0, NULL, 0))
00265
00266
                 free (sqe);
                 nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
res = ERROR_CODE;
00267
00268
00269
                 continue;
00270
00271
            req->client_socket = 0;
00272
            req->event_type = EV_ACC;
00273
             io_uring_prep_accept (sqe, s->ss, (struct sockaddr *const)ca, cal, 0);
00274
             io_uring_sqe_set_data (sqe, req);
00275
            if (io_uring_submit (&s->rings[0]) < 0)</pre>
00276
              {
00277
                 free (sqe);
                 list_delete_node (&s->list, req);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00278
00279
                 res = ERROR_CODE;
00280
00281
                 continue;
00282
00283
            res = SUCCESS_CODE;
00284
00285
       pthread_mutex_unlock (&s->m_rings[0]);
00286 }
```

#### 7.13.2.2 add\_efd()

```
static void add_efd (
               struct saurion *const s,
                const int client_socket,
                int sel ) [static]
Definition at line 289 of file low_saurion.c.
00290
00291
        pthread_mutex_lock (&s->m_rings[sel]);
00292
        int res = ERROR_CODE;
00293
        while (res != SUCCESS_CODE)
00294
00295
             struct io_uring *ring = &s->rings[sel];
00296
             struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00297
            while (!sqe)
00298
              {
00299
                sqe = io_uring_get_sqe (ring);
00300
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00301
            if (!set_request *req = NULL;
if (!set_request (&req, &s->list, CHUNK_SZ, NULL, 0))
00302
00303
00304
              {
00305
                 free (sqe);
res = ERROR_CODE;
00306
00307
                 continue;
00308
            req->event_type = EV_REA;
req->client_socket = client_socket;
00309
00310
             io_uring_prep_readv (sqe, client_socket, &req->iov[0], req->iovec_count,
00311
00312
                                   0);
00313
             io_uring_sqe_set_data (sqe, req);
00314
             if (io_uring_submit (ring) < 0)</pre>
00315
              {
00316
                 free (sqe);
                 list_delete_node (&s->list, req);
00317
                 res = ERROR_CODE;
00318
00319
                 continue;
00320
00321
             res = SUCCESS_CODE;
00322
        pthread_mutex_unlock (&s->m_rings[sel]);
00323
00324 }
```

#### 7.13.2.3 add\_read()

### Definition at line 327 of file low\_saurion.c.

```
00328 {
00329
        int sel = next (s);
00330
        int res = ERROR_CODE;
00331
        pthread_mutex_lock (&s->m_rings[sel]);
00332
        while (res != SUCCESS_CODE)
00333
00334
            struct io_uring *ring = &s->rings[sel];
00335
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00336
            while (!sge)
00337
             {
00338
                sqe = io_uring_get_sqe (ring);
00339
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00340
            struct request *req = NULL;
00341
00342
            if (!set_request (&req, &s->list, CHUNK_SZ, NULL, 0))
00343
00344
                free (sqe);
00345
                res = ERROR_CODE;
00346
                continue;
00347
00348
            req->event_type = EV_REA;
00349
            req->client_socket = client_socket;
```

```
00350
            io_uring_prep_readv (sqe, client_socket, &req->iov[0], req->iovec_count,
00351
00352
             io_uring_sqe_set_data (sqe, req);
00353
            if (io_uring_submit (ring) < 0)</pre>
00354
                 free (sqe);
list_delete_node (&s->list, req);
00355
00356
00357
                 res = ERROR_CODE;
                continue;
00358
00359
00360
             res = SUCCESS_CODE;
00362 pthread_mutex_unlock (&s->m_rings[sel]);
00363 }
00361
```

#### 7.13.2.4 add\_read\_continue()

```
static void add_read_continue (
    struct saurion *const s,
    struct request * oreq,
    const int sel ) [static]
```

#### Definition at line 366 of file low\_saurion.c.

```
00369
        pthread_mutex_lock (&s->m_rings[sel]);
00370
        int res = ERROR_CODE;
00371
        while (res != SUCCESS_CODE)
00372
00373
            struct io_uring *ring = &s->rings[sel];
00374
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
            while (!sqe)
00375
00376
             {
                sqe = io_uring_get_sqe (ring);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00377
00378
00379
00380
            if (!set_request (&oreq, &s->list, oreq->prev_remain, NULL, 0))
00381
00382
                free (sqe);
00383
                res = ERROR_CODE;
00384
                continue;
00385
00386
            io_uring_prep_readv (sqe, oreq->client_socket, &oreq->iov[0],
00387
                                  oreq->iovec_count, 0);
            io_uring_sqe_set_data (sqe, oreq);
00388
00389
            if (io_uring_submit (ring) < 0)</pre>
00390
              {
00391
                free (sqe);
00392
                list_delete_node (&s->list, oreq);
00393
                res = ERROR_CODE;
00394
                continue;
00395
00396
            res = SUCCESS_CODE;
00397
       pthread_mutex_unlock (&s->m_rings[sel]);
00399 }
```

#### 7.13.2.5 add\_write()

#### Definition at line 402 of file low saurion.c.

```
00404 {
00405 int res = ERROR_CODE;
```

```
pthread_mutex_lock (&s->m_rings[sel]);
00407
        while (res != SUCCESS_CODE)
00408
00409
            struct io_uring *ring = &s->rings[sel];
00410
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00411
            while (!sqe)
00412
             {
00413
                sqe = io_uring_get_sqe (ring);
00414
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00415
            struct request *req = NULL;
00416
            if (!set_request (&req, &s->list, strlen (str), (const void *const)str,
00417
00418
                              1))
00419
00420
                free (sqe);
00421
                res = ERROR_CODE;
00422
                continue;
              }
00423
00424
           req->event_type = EV_WRI;
00425
            req->client_socket = fd;
00426
            io_uring_prep_writev (sqe, req->client_socket, req->iov,
00427
                                  req->iovec_count, 0);
            io_uring_sqe_set_data (sqe, req);
00428
00429
            if (io_uring_submit (ring) < 0)</pre>
00430
              {
00431
                free (sqe);
00432
                list_delete_node (&s->list, req);
00433
                res = ERROR_CODE;
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00434
00435
                continue;
00436
00437
            res = SUCCESS_CODE;
00438
00439
       pthread_mutex_unlock (&s->m_rings[sel]);
00440 }
```

#### 7.13.2.6 handle\_accept()

#### Definition at line 444 of file low saurion.c.

# 7.13.2.7 handle\_close()

#### Definition at line 685 of file low saurion.c.

#### 7.13.2.8 handle\_error()

```
static void handle_error (
              const struct saurion *const s,
              const struct request *const req ) [static]
Definition at line 674 of file low_saurion.c.
00675 {
00676
        if (s->cb.on_error)
00677
00678
           const char *resp = "ERROR";
00679
           s->cb.on_error (req->client_socket, resp, (ssize_t)strlen (resp),
00680
                           s->cb.on_error_arg);
00681
         }
00682 }
```

#### 7.13.2.9 handle\_read()

#### Definition at line 627 of file low saurion.c.

```
00629
        void *msg = NULL;
00630
        size_t len = 0;
00631
        while (1)
00632
            if (!read_chunk (&msg, &len, req))
00633
00634
               break;
00636
00637
            if (req->next_iov || req->next_offset)
00638
00639
               if (s->cb.on_readed && msg)
00640
                {
00641
                   s->cb.on_readed (req->client_socket, msg, len,
00642
                                    s->cb.on_readed_arg);
00643
00644
               free (msg);
               msg = NULL;
00645
00646
               continue;
00647
00648
            if (req->prev && req->prev_size && req->prev_remain)
00649
            {
00650
               add_read_continue (s, req, next (s));
00651
                return;
00652
00653
            if (s->cb.on_readed && msq)
           {
00655
               s->cb.on_readed (req->client_socket, msg, len, s->cb.on_readed_arg);
00656
00657
           free (msg);
00658
            msq = NULL;
00659
            break;
00660
00661
       add_read (s, req->client_socket);
00662 }
```

#### 7.13.2.10 handle\_write()

Definition at line 665 of file low\_saurion.c.

#### 7.13.2.11 htonll()

#### Definition at line 65 of file low saurion.c.

#### 7.13.2.12 next()

```
static uint32_t next ( {\tt struct \ saurion * s )} \quad [{\tt static}]
```

#### Definition at line 58 of file low\_saurion.c.

```
00059 {
00060    s->next = (s->next + 1) % s->n_threads;
00061    return s->next;
00062 }
```

#### 7.13.2.13 ntohll()

#### Definition at line 78 of file low\_saurion.c.

```
00079 {
00080    int num = 42;
00081    if (*(char *)&num == 42)
00082    {
00083         uint32_t high_part = ntohl ((uint32_t) (value » 32));
00084         uint32_t low_part = ntohl ((uint32_t) (value & 0xFFFFFFFLL));
00085         return ((uint64_t)low_part « 32) | high_part;
00086    }
00087    return value;
00088 }
```

#### 7.13.2.14 saurion\_worker\_master()

```
void saurion_worker_master (
              void * arg )
Definition at line 917 of file low_saurion.c.
00918 {
        LOG_INIT (" ");
00919
        struct saurion *const s = (struct saurion *)arg;
struct sockaddr_in client_addr;
00920
00922
        socklen_t client_addr_len = sizeof (client_addr);
00923
00924
        add_efd (s, s->efds[0], 0);
00925
       add_accept (s, &client_addr, &client_addr_len);
00926
00927
        pthread_mutex_lock (&s->status_m);
00928
        ++s->status;
00929
        pthread_cond_broadcast (&s->status_c);
00930
        pthread_mutex_unlock (&s->status_m);
00931
        while (1)
00932
         {
00933
            int ret
00934
                 = saurion_worker_master_loop_it (s, &client_addr, &client_addr_len);
00935
            if (ret == ERROR_CODE || ret == CRITICAL_CODE)
00936
              {
00937
                break;
00938
              }
00939
00940 pthread_mutex_lock (&s->status_m);
00941
         -s->status;
00942
        pthread_cond_signal (&s->status_c);
        pthread_mutex_unlock (&s->status_m);
LOG_END (" ");
00943
00944
00945
        return:
00946 }
```

#### 7.13.2.15 saurion\_worker\_master\_loop\_it()

#### Definition at line 849 of file low\_saurion.c.

```
00852
         LOG_INIT (" ");
00853
         struct io_uring ring = s->rings[0];
struct io_uring_cqe *cqe = NULL;
00854
00855
00856
         int ret = io_uring_wait_cqe (&ring, &cqe);
         if (ret < 0)
00857
00858
          {
             free (cqe);
LOG_END (" ");
00859
00860
             return CRITICAL_CODE;
00861
00862
00863
         struct request *req = (struct request *)cqe->user_data;
00864
         if (!req)
00865
             io_uring_cqe_seen (&s->rings[0], cqe);
LOG_END (" ");
00866
00867
             return SUCCESS_CODE;
00868
00869
00870
         if (cqe->res < 0)</pre>
00871
             list_delete_node (&s->list, req);
LOG_END (" ");
00872
00873
00874
             return CRITICAL_CODE;
00875
00876
         if (req->client_socket == s->efds[0])
00877
00878
             io_uring_cqe_seen (&s->rings[0], cqe);
             list_delete_node (&s->list, req);
LOG_END (" ");
00879
00880
             return ERROR_CODE;
```

```
00883
        io_uring_cqe_seen (&s->rings[0], cqe);
00884
        switch (req->event_type)
00885
          case EV_ACC:
00886
00887
           handle_accept (s, cqe->res);
            add_accept (s, client_addr, client_addr_len);
00889
            add_read (s, cqe->res);
00890
            list_delete_node (&s->list, req);
          break;
case EV_REA:
00891
00892
00893
           if (cqe->res < 0)</pre>
00894
             {
00895
                handle_error (s, req);
00896
00897
            if (cqe->res < 1)</pre>
00898
00899
                handle_close (s, req);
00900
00901
            if (cqe->res > 0)
00902
00903
                handle_read (s, req);
00904
00905
            list_delete_node (&s->list, req);
00906
            break;
00907
          case EV_WRI:
00908
            handle_write (s, req->client_socket);
00909
            list_delete_node (&s->list, req);
00910
            break;
00911
       LOG_END (" ");
00912
00913
       return SUCCESS_CODE;
00914 }
```

#### 7.13.2.16 saurion\_worker\_slave()

```
void saurion_worker_slave (
     void * arg )
```

### Definition at line 1012 of file low\_saurion.c.

```
01013 {
01014
        LOG_INIT (" ");
01015
        struct saurion_wrapper *const ss = (struct saurion_wrapper *)arg;
01016
        struct saurion *s = ss->s;
01017
        const int sel = ss->sel;
01018
       free (ss);
01019
01020
       add_efd (s, s->efds[sel], sel);
01022
       pthread_mutex_lock (&s->status_m);
01023
        ++s->status;
        pthread_cond_broadcast (&s->status_c);
01024
01025
        pthread_mutex_unlock (&s->status_m);
01026
        while (1)
01027
         {
01028
            int res = saurion_worker_slave_loop_it (s, sel);
01029
            if (res == ERROR_CODE || res == CRITICAL_CODE)
01030
01031
                break:
              }
01032
01033
01034
        pthread_mutex_lock (&s->status_m);
01035
         --s->status;
01036
        pthread_cond_signal (&s->status_c);
        pthread_mutex_unlock (&s->status_m);
LOG_END (" ");
01037
01038
01039
        return;
01040 }
```

#### 7.13.2.17 saurion\_worker\_slave\_loop\_it()

```
static int saurion_worker_slave_loop_it (
                struct saurion *const s,
                const int sel ) [static]
Definition at line 950 of file low_saurion.c.
00951 {
         LOG_INIT (" ");
00952
        struct io_uring ring = s->rings[sel];
struct io_uring_cqe *cqe = NULL;
00953
00954
00955
00956
        add_efd (s, s->efds[sel], sel);
        int ret = io_uring_wait_cqe (&ring, &cqe);
if (ret < 0)</pre>
00957
00958
00959
          {
             free (cqe);
LOG_END (" ");
00960
00961
00962
             return CRITICAL_CODE;
00963
00964
        struct request *req = (struct request *)cqe->user_data;
00965
        if (!req)
00966
            io_uring_cqe_seen (&ring, cqe);
LOG_END (" ");
00967
00968
00969
             return SUCCESS_CODE;
00970
00971
         if (cqe->res < 0)</pre>
00972
         {
             list_delete_node (&s->list, req);
LOG_END (" ");
00973
00974
00975
             return CRITICAL_CODE;
00976
00977
        if (req->client_socket == s->efds[sel])
00978
00979
             io_uring_cqe_seen (&ring, cqe);
             list_delete_node (&s->list, req);
LOG_END (" ");
00981
             return ERROR_CODE;
00982
00983
00984
        io_uring_cqe_seen (&ring, cqe);
switch (req->event_type)
00985
         {
case EV_REA:
00986
00987
00988
             if (cqe->res < 0)</pre>
00989
              {
00990
                 handle_error (s, req);
00991
00992
             if (cqe->res < 1)</pre>
00993
              {
00994
                 handle_close (s, req);
00995
00996
             if (cqe->res > 0)
00997
00998
                 handle_read (s, req);
01000
             list_delete_node (&s->list, req);
01001
            break;
          case EV_WRI:
01002
            handle_write (s, req->client_socket);
list_delete_node (&s->list, req);
01003
01004
01005
             break;
01006
        LOG_END (" ");
01007
        return SUCCESS_CODE;
01008
01009 }
```

#### 7.13.3 Variable Documentation

#### 7.13.3.1 print\_mutex

pthread\_mutex\_t print\_mutex

Definition at line 47 of file low saurion.c.

#### 7.13.3.2 TIMEOUT\_RETRY\_SPEC

```
struct timespec TIMEOUT_RETRY_SPEC = { 0, TIMEOUT_RETRY * 1000L }
```

Definition at line 49 of file low saurion.c.

# 7.14 low saurion.c

#### Go to the documentation of this file.

```
00001 #include "low_saurion.h" 00002 #include "config.h"
00002 #include "config.h" // for ERROR_CODE, SUCCESS_CODE, CHUNK_SZ
00003 #include "linked_list.h" // for list_delete_node, list_free, list_insert
00004 #include "threadpool.h" // for threadpool_add, threadpool_create
00006 #include <arpa/inet.h>
                                                              // for htonl, ntohl, htons
00007 #include <bits/socket-constants.h> // for SOL_SOCKET, SO_REUSEADDR
00008 #include <liburing.h> // for io_uring_get_sqe, io_uring, io_uring_...
00009 #include <liburing/io_uring.h> // for io_uring_cqe
UUUU9 #include Iiburing/lo_uring.h> // for io_uring_cqe
00010 #include <nanologger.h> // for LOG_END, LOG_INIT
00011 #include <netinet/in.h> // for sockaddr_in, INADDR_ANY, in_addr
00012 #include <pthread.h> // for pthread_mutex_lock, pthread_mutex_unlock
00013 #include <stdio.h> // for uint32_t, uint64_t, uint8_t
00014 #include <stdio.h> // for NULL
00015 #include <stdib.h> // for free, malloc
00016 #include <string.h> // for memset. memory. strlen
00022
00023 struct Node;
00024 struct iovec;
00025
00026 #define EV_ACC 0
00027 #define EV_REA 1
00028 #define EV_WRI 2
00029 #define EV_WAI
00030 #define EV_ERR 4
00031
00032 struct request
00033 {
00034 void *prev;
00035 size_t prev_size;
 00036
           size_t prev_remain;
00037
           size_t next_iov;
00038
           size_t next_offset;
00039
           int event_type;
00040
           size t iovec count;
00041
           int client_socket;
00042
           struct iovec iov[];
00043 };
00044
00045 #define MIN(a, b) ((a) < (b) ? (a) : (b))
00046 #define MAX(a, b) ((a) > (b) ? (a) : (b))
00047 pthread_mutex_t print_mutex;
00048
00049 struct timespec TIMEOUT_RETRY_SPEC = { 0, TIMEOUT_RETRY * 1000L };
00050
00051 struct saurion_wrapper
00052 {
00053
           struct saurion *s;
           uint32_t sel;
00055 };
00056
00057 static uint32_t
00058 next (struct saurion *s)
00059 {
00060
           s \rightarrow next = (s \rightarrow next + 1) % s \rightarrow n_threads;
           return s->next;
00062 }
00063
00064 static uint64_t
00065 htonll (uint64_t value)
00066 {
            int num = 42;
00068
         if (*(char *)&num == 42)
00069
```

7.14 low\_saurion.c 83

```
uint32_t high_part = htonl ((uint32_t) (value » 32));
00071
            uint32_t low_part = htonl ((uint32_t) (value & 0xFFFFFFFFLL));
00072
            return ((uint64_t)low_part « 32) | high_part;
00073
00074
        return value;
00075 }
00076
00077 static uint64_t
00078 ntohll (uint64_t value)
00079 {
08000
        int num = 42:
00081
        if (*(char *)&num == 42)
         {
00082
00083
            uint32_t high_part = ntohl ((uint32_t) (value » 32));
00084
            uint32_t low_part = ntohl ((uint32_t) (value & 0xfffffffffLL));
00085
            return ((uint64_t)low_part « 32) | high_part;
00086
00087
       return value;
00088 }
00089
00090 void
00091 free_request (struct request *req, void **children_ptr, size_t amount)
00092 {
00093
        if (children ptr)
00094
         {
00095
            free (children_ptr);
00096
            children_ptr = NULL;
00097
        for (size_t i = 0; i < amount; ++i)</pre>
00098
00099
00100
           free (req->iov[i].iov_base);
00101
           req->iov[i].iov_base = NULL;
00102
00103
       free (req);
00104
       req = NULL;
       free (children_ptr);
00105
00106
       children_ptr = NULL;
00108
00109 [[nodiscard]]
00110 int
00111 initialize_iovec (struct iovec \stariov, size_t amount, size_t pos,
00112
                         const void *msg, size_t size, uint8_t h)
00113 {
00114
       if (!iov || !iov->iov_base)
00115
00116
            return ERROR_CODE;
00117
       if (msg)
00118
00119
00120
            size_t len = iov->iov_len;
            char *dest = (char *)iov->iov_base;
char *orig = (char *)msg + pos * CHUNK_SZ;
00121
00122
00123
            size_t cpy_sz = 0;
00124
            if (h)
00125
              {
                if (pos == 0)
00127
                  {
00128
                    uint64_t send_size = htonll (size);
                    memcpy (dest, &send_size, sizeof (uint64_t));
dest += sizeof (uint64_t);
00129
00130
                    len -= sizeof (uint64_t);
00131
00132
00133
00134
00135
                    orig -= sizeof (uint64_t);
00136
                if ((pos + 1) == amount)
00137
00138
00139
                     --len;
00140
                    cpy_sz = (len < size ? len : size);</pre>
00141
                    dest[cpy_sz] = 0;
00142
                  }
             }
00143
00144
            cpy_sz = (len < size ? len : size);</pre>
00145
            memcpy (dest, orig, cpy_sz);
00146
            dest += cpy_sz;
00147
            size_t rem = CHUNK_SZ - (dest - (char *)iov->iov_base);
00148
            memset (dest, 0, rem);
00149
00150
        else
00151
        {
00152
            memset ((char *)iov->iov_base, 0, CHUNK_SZ);
00153
00154
       return SUCCESS_CODE;
00155 }
00156
```

```
00157 [[nodiscard]]
00158 int
00159 allocate_iovec (struct iovec *iov, size_t amount, size_t pos, size_t size,
00160
                       void **chd_ptr)
00161 {
00162
        if (!iov || !chd ptr)
00163
00164
            return ERROR_CODE;
00165
00166
        iov->iov_base = malloc (CHUNK_SZ);
        if (!iov->iov_base)
00167
00168
         {
00169
            return ERROR CODE;
00170
00171
        iov->iov_len = (pos == (amount - 1) ? (size % CHUNK_SZ) : CHUNK_SZ);
00172
        if (iov->iov_len == 0)
00173
00174
            iov->iov len = CHUNK SZ;
00176
       chd_ptr[pos] = iov->iov_base;
       return SUCCESS_CODE;
00177
00178 }
00179
00180 [[nodiscard]]
00181 int
00182 set_request (struct request **r, struct Node **1, size_t s, const void *m,
00183
                   uint8_t h)
00184 {
00185
       uint64_t full_size = s;
00186
        if (h)
00187
00188
            full_size += (sizeof (uint64_t) + sizeof (uint8_t));
00189
00190
       size_t amount = full_size / CHUNK_SZ;
        amount = amount + (full_size % CHUNK_SZ == 0 ? 0 : 1);
struct request *temp = (struct request *)malloc (
00191
00192
            sizeof (struct request) + sizeof (struct iovec) * amount);
00193
00194
        if (!temp)
00195
        {
00196
            return ERROR_CODE;
00197
          }
        if (!*r)
00198
00199
         {
00200
            *r = temp;
00201
           (*r) \rightarrow prev = NULL;
00202
            (*r) \rightarrow prev\_size = 0;
00203
            (*r)->prev_remain = 0;
00204
            (*r) \rightarrow next_iov = 0;
00205
            (*r) \rightarrow next_offset = 0;
00206
00207
       else
00208
        {
00209
            temp->client_socket = (*r)->client_socket;
00210
            temp->event_type = (*r)->event_type;
            temp->prev = (*r)->prev;
00211
00212
            temp->prev_size = (*r)->prev_size;
00213
            temp->prev_remain = (*r)->prev_remain;
00214
            temp->next_iov = (*r)->next_iov;
00215
            temp->next_offset = (*r)->next_offset;
00216
            *r = temp;
00217
00218
       struct request *req = *r;
00219
        req->iovec_count = (int)amount;
00220
        void **children_ptr = (void **)malloc (amount * sizeof (void *));
00221
        if (!children_ptr)
00222
         {
            free_request (req, children_ptr, 0);
00223
00224
            return ERROR_CODE;
00225
00226
        for (size_t i = 0; i < amount; ++i)</pre>
00227
00228
            if (!allocate_iovec (&req->iov[i], amount, i, full_size, children_ptr))
00229
             {
00230
                free_request (req, children_ptr, amount);
00231
                return ERROR_CODE;
00232
00233
            if (!initialize_iovec (&req->iov[i], amount, i, m, s, h))
00234
                free_request (req, children_ptr, amount);
00235
                return ERROR CODE:
00236
00237
00238
00239
        if (list_insert (l, req, amount, children_ptr))
00240
00241
            free_request (req, children_ptr, amount);
00242
            return ERROR_CODE;
00243
          }
```

7.14 low\_saurion.c 85

```
free (children_ptr);
       return SUCCESS_CODE;
00245
00246 }
00247
00248 /********** ADDERS ************
00249 static void
00250 add_accept (struct saurion *const s, struct sockaddr_in *const ca,
00251
                   socklen_t *const cal)
00252 {
        int res = ERROR_CODE;
00253
        pthread_mutex_lock (&s->m_rings[0]);
while (res != SUCCESS_CODE)
00254
00255
00256
          {
00257
            struct io_uring_sqe *sqe = io_uring_get_sqe (&s->rings[0]);
00258
             while (!sqe)
00259
                sqe = io_uring_get_sqe (&s->rings[0]);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00260
00261
00262
00263
            struct request *req = NULL;
00264
             if (!set_request (&req, &s->list, 0, NULL, 0))
00265
               {
00266
                 free (sqe);
                 nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00267
00268
                 res = ERROR_CODE;
                 continue;
00269
00270
00271
            req->client_socket = 0;
00272
             req->event_type = EV_ACC;
00273
             io_uring_prep_accept (sqe, s->ss, (struct sockaddr \starconst)ca, cal, 0);
00274
             io_uring_sqe_set_data (sqe, req);
00275
             if (io_uring_submit (&s->rings[0]) < 0)</pre>
00276
00277
                 free (sqe);
                 list_delete_node (&s->list, req);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00278
00279
                 res = ERROR_CODE;
00280
                 continue;
00281
00282
00283
             res = SUCCESS_CODE;
00284
00285
        pthread_mutex_unlock (&s->m_rings[0]);
00286 }
00287
00288 static void
00289 add_efd (struct saurion *const s, const int client_socket, int sel)
00290 {
00291
        pthread_mutex_lock (&s->m_rings[sel]);
        int res = ERROR_CODE;
00292
        while (res != SUCCESS_CODE)
00293
00294
          {
00295
            struct io_uring *ring = &s->rings[sel];
00296
             struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00297
             while (!sqe)
00298
              {
00299
                 sqe = io_uring_get_sqe (ring);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00300
00301
00302
             struct request *req = NULL;
             if (!set_request (&req, &s->list, CHUNK_SZ, NULL, 0))
00303
00304
              {
                free (sqe);
res = ERROR_CODE;
00305
00306
00307
                 continue;
00308
00309
            req->event_type = EV_REA;
00310
             req->client_socket = client_socket;
             io_uring_prep_readv (sqe, client_socket, &req->iov[0], req->iovec_count,
00311
00312
                                   0);
00313
             io_uring_sqe_set_data (sqe, req);
00314
             if (io_uring_submit (ring) < 0)</pre>
00315
00316
                 free (sqe);
                 list_delete_node (&s->list, req);
res = ERROR_CODE;
00317
00318
00319
                 continue;
00320
00321
             res = SUCCESS_CODE;
00322
00323
        pthread_mutex_unlock (&s->m_rings[sel]);
00324 }
00325
00326 static void
00327 add_read (struct saurion *const s, const int client_socket)
00328 {
00329
        int sel = next (s);
00330
       int res = ERROR_CODE;
```

```
pthread_mutex_lock (&s->m_rings[sel]);
        while (res != SUCCESS_CODE)
00332
00333
00334
            struct io_uring *ring = &s->rings[sel];
00335
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00336
            while (!sqe)
             {
00338
                sqe = io_uring_get_sqe (ring);
00339
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00340
00341
            struct request *req = NULL;
            if (!set_request (&req, &s->list, CHUNK_SZ, NULL, 0))
00342
00343
             {
00344
                free (sqe);
00345
                res = ERROR_CODE;
                continue;
00346
00347
00348
            req->event_type = EV_REA;
            req->client_socket = client_socket;
00349
00350
            io_uring_prep_readv (sqe, client_socket, &req->iov[0], req->iovec_count,
00351
                                 0);
00352
            io_uring_sqe_set_data (sqe, req);
00353
            if (io_uring_submit (ring) < 0)</pre>
00354
              {
00355
                free (sqe);
                list_delete_node (&s->list, req);
00356
00357
                res = ERROR_CODE;
00358
                continue;
00359
00360
           res = SUCCESS CODE;
00361
00362
       pthread_mutex_unlock (&s->m_rings[sel]);
00363 }
00364
00365 static void
00366 add_read_continue (struct saurion *const s, struct request *oreq,
00367
                         const int sel)
00368 {
00369
       pthread_mutex_lock (&s->m_rings[sel]);
00370
        int res = ERROR_CODE;
00371
        while (res != SUCCESS_CODE)
00372
           struct io_uring *ring = &s->rings[sel];
00373
00374
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00375
            while (!sqe)
00376
             {
00377
                sqe = io_uring_get_sqe (ring);
00378
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00379
00380
            if (!set request (&oreg, &s->list, oreg->prev remain, NULL, 0))
00381
             {
00382
                free (sqe);
00383
                res = ERROR_CODE;
00384
                continue;
00385
00386
           io_uring_prep_readv (sqe, oreq->client_socket, &oreq->iov[0],
                                 oreq->iovec_count, 0);
00388
            io_uring_sqe_set_data (sqe, oreq);
00389
            if (io_uring_submit (ring) < 0)</pre>
00390
00391
                free (sge);
00392
                list_delete_node (&s->list, oreq);
00393
                res = ERROR_CODE;
00394
                continue;
00395
00396
            res = SUCCESS_CODE;
00397
       pthread_mutex_unlock (&s->m_rings[sel]);
00398
00399 }
00401 static void
00402 add_write (struct saurion *const s, int fd, const char *const str,
00403
                 const int sel)
00404 {
00405
       int res = ERROR_CODE;
        pthread_mutex_lock (&s->m_rings[sel]);
00406
00407
        while (res != SUCCESS_CODE)
00408
00409
            struct io_uring *ring = &s->rings[sel];
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00410
00411
            while (!sge)
00412
             {
00413
                sqe = io_uring_get_sqe (ring);
00414
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00415
            struct request *req = NULL;
00416
            if (!set_request (&req, &s->list, strlen (str), (const void *const)str,
00417
```

7.14 low\_saurion.c 87

```
00418
                               1))
00419
00420
                free (sqe);
00421
                res = ERROR CODE;
00422
                continue;
00423
              }
            req->event_type = EV_WRI;
00424
00425
             req->client_socket = fd;
00426
            io_uring_prep_writev (sqe, req->client_socket, req->iov,
00427
                                    req->iovec_count, 0);
            io_uring_sqe_set_data (sqe, req);
00428
00429
            if (io_uring_submit (ring) < 0)</pre>
00430
              {
00431
                 free (sqe);
00432
                 list_delete_node (&s->list, req);
00433
                 res = ERROR_CODE;
                 nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00434
00435
                continue;
00436
            res = SUCCESS_CODE;
00437
00438
00439
       pthread_mutex_unlock (&s->m_rings[sel]);
00440 }
00441
00442 /************** HANDLERS *****************
00443 static void
00444 handle_accept (const struct saurion *const s, const int fd)
00445 {
00446
        if (s->cb.on_connected)
00447
        {
00448
            s->cb.on_connected (fd, s->cb.on_connected_arg);
00449
00450 }
00451
00452 [[nodiscard]]
00453 int
00454 read_chunk (void **dest, size_t *len, struct request *const req)
00456
        if (req->iovec_count == 0)
00457
        {
00458
            return ERROR_CODE;
         }
00459
00460
00461
        size_t max_iov_cont = 0; //< Total size of request</pre>
        for (size_t i = 0; i < req->iovec_count; ++i)
00463
00464
            max_iov_cont += req->iov[i].iov_len;
00465
00466
        size t cont sz = 0;
00467
        size t cont rem = 0;
        size_t curr_iov = 0;
00468
00469
        size_t curr_iov_off = 0;
        size_t dest_off = 0;
void *dest_ptr = NULL;
00470
00471
00472
        if (req->prev && req->prev_size && req->prev_remain)
00473
        {
            cont_sz = req->prev_size;
00475
            cont_rem = req->prev_remain;
            curr_iov = 0;
curr_iov_off = 0;
00476
00477
00478
            dest_off = cont_sz - cont_rem;
00479
            if (cont_rem <= max_iov_cont)</pre>
00480
              {
00481
               *dest = req->prev;
00482
                dest_ptr = *dest;
                req->prev = NULL;
00483
00484
                req->prev_size = 0;
                req->prev_remain = 0;
00485
00486
00487
            else
00488
             {
00489
                dest_ptr = req->prev;
00490
                *dest = NULL;
00491
00492
00493
        else if (req->next_iov || req->next_offset)
00494
         {
00495
            curr_iov = req->next_iov;
00496
            curr_iov_off = req->next_offset;
00497
            cont_sz = *(
00498
               (size_t *)(((uint8_t *)req->iov[curr_iov].iov_base) + curr_iov_off));
00499
            cont_sz = ntohll (cont_sz);
00500
            curr_iov_off += sizeof (uint64_t);
            cont_rem = cont_sz;
dest_off = cont_sz - cont_rem;
if ((curr_iov_off + cont_rem + 1) <= max_iov_cont)</pre>
00501
00502
00503
00504
```

```
*dest = malloc (cont_sz);
00506
               dest_ptr = *dest;
00507
00508
           else
00509
             {
00510
               req->prev = malloc (cont_sz);
00511
               dest_ptr = req->prev;
00512
               *dest = NULL;
00513
               *len = 0;
00514
             }
00515
         }
00516
       else
00517
        {
00518
           curr_iov = 0;
00519
           curr_iov_off = 0;
00520
            cont_sz = *(
                (size_t *)(((uint8_t *)req->iov[curr_iov].iov_base) + curr_iov_off));
00521
           cont_sz = ntohll (cont_sz);
curr_iov_off += sizeof (uint64_t);
00522
00524
            cont_rem = cont_sz;
00525
            dest_off = cont_sz - cont_rem;
00526
            if (cont_rem <= max_iov_cont)</pre>
00527
             {
                *dest = malloc (cont_sz);
00528
00529
               dest_ptr = *dest;
00530
00531
            else
00532
             {
00533
               req->prev = malloc (cont_sz);
                dest_ptr = req->prev;
00534
00535
               *dest = NULL;
00536
00537
00538
       size_t curr_iov_msg_rem = 0;
00539
       uint8_t ok = 1UL;
00540
00541
       while (1)
00542
00543
           curr_iov_msg_rem
00544
                = MIN (cont_rem, (req->iov[curr_iov].iov_len - curr_iov_off));
           00545
00546
00547
                    curr_iov_msg_rem);
           dest_off += curr_iov_msg_rem;
00548
00549
           curr_iov_off += curr_iov_msg_rem;
            cont_rem -= curr_iov_msg_rem;
00550
00551
            if (cont_rem <= 0)</pre>
00552
00553
                if (*(((uint8_t *)req->iov[curr_iov].iov_base) + curr_iov_off) != 0)
00554
                 {
                   ok = OUL;
00556
                *len = cont_sz;
00557
00558
                ++curr_iov_off;
00559
               break:
00560
              (curr_iov_off >= (req->iov[curr_iov].iov_len))
00562
              {
00563
                ++curr_iov;
00564
                if (curr_iov == req->iovec_count)
00565
00566
                   break;
00567
00568
               curr_iov_off = 0;
00569
00570
         }
00571
00572
       if (reg->prev)
00573
00574
           req->prev_size = cont_sz;
00575
            req->prev_remain = cont_rem;
00576
            *dest = NULL;
           len = 0;
00577
00578
00579
       else
00580
        {
00581
            req->prev_size = 0;
00582
           req->prev_remain = 0;
00583
       if (curr_iov < req->iovec_count)
00584
00585
           uint64_t next_sz = *(uint64_t *)(((uint8_t *)req->iov[curr_iov].iov_base)
00587
                                             + curr_iov_off);
            if ((req->iov[curr_iov].iov_len > curr_iov_off) && next_sz)
00588
00589
               req->next_iov = curr_iov;
00590
                req->next_offset = curr_iov_off;
00591
```

7.14 low\_saurion.c 89

```
00592
00593
00594
             {
00595
               req->next_iov = 0;
               req->next_offset = 0;
00596
00597
00598
         }
00599
00600
       if (ok)
00601
           return SUCCESS_CODE;
00602
00603
        free (dest_ptr);
00604
00605
        dest_ptr = NULL;
00606
        *dest = NULL;
        *len = 0;
00607
        req->next_iov = 0;
00608
        req->next_offset = 0;
00609
00610
        for (size_t i = curr_iov; i < req->iovec_count; ++i)
00611
         {
00612
            for (size_t j = curr_iov_off; j < req->iov[i].iov_len; ++j)
00613
               uint8_t foot = *((uint8_t *)req->iov[i].iov_base) + j;
00614
00615
                if (foot == 0)
00616
                 {
00617
                   req->next_iov = i;
00618
                    req->next_offset = (j + 1) % req->iov[i].iov_len;
00619
                    return ERROR_CODE;
00620
00621
             }
00622
00623
       return ERROR_CODE;
00624 }
00625
00626 static void
00627 handle_read (struct saurion *const s, struct request *const req)
00628 {
00629
       void *msg = NULL;
00630
       size_t len = 0;
00631
        while (1)
00632
00633
            if (!read_chunk (&msg, &len, req))
00634
00635
                break;
00636
00637
            if (req->next_iov || req->next_offset)
00638
00639
                if (s->cb.on_readed && msg)
00640
                 {
00641
                   s->cb.on_readed (req->client_socket, msg, len,
00642
                                     s->cb.on_readed_arg);
00643
00644
                free (msg);
00645
                msg = NULL;
00646
                continue;
00647
00648
            if (req->prev && req->prev_size && req->prev_remain)
00649
             {
00650
               add_read_continue (s, req, next (s));
00651
                return;
00652
00653
            if (s->cb.on_readed && msg)
00654
             {
00655
               s->cb.on_readed (req->client_socket, msg, len, s->cb.on_readed_arg);
00656
00657
            free (msg);
            msg = NULL;
00658
00659
           break:
00660
00661
       add_read (s, req->client_socket);
00662 }
00663
00664 static void
00665 handle_write (const struct saurion *const s, const int fd)
00666 {
00667
       if (s->cb.on_wrote)
00668
        {
00669
            s->cb.on_wrote (fd, s->cb.on_wrote_arg);
00670
00671 }
00672
00673 static void
00674 handle_error (const struct saurion *const s, const struct request *const req)
00675 {
00676
        if (s->cb.on_error)
00677
00678
            const char *resp = "ERROR";
```

```
s->cb.on_error (req->client_socket, resp, (ssize_t)strlen (resp),
00680
                             s->cb.on_error_arg);
00681
          }
00682 }
00683
00684 static void
00685 handle_close (const struct saurion *const s, const struct request *const req)
00686 {
00687
        if (s->cb.on_closed)
00688
            s->cb.on_closed (req->client_socket, s->cb.on_closed_arg);
00689
00690
00691
       close (req->client_socket);
00692 }
00693
00695 int
00696 EXTERNAL set socket (const int p)
00697 {
00698
        int sock = 0;
00699
        struct sockaddr_in srv_addr;
00700
00701
        sock = socket (PF_INET, SOCK_STREAM, 0);
00702
        if (sock < 1)
00703
        {
00704
            return ERROR_CODE;
00705
00706
00707
        int enable = 1;
        if (setsockopt (sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof (int)) < 0)</pre>
00708
00709
00710
            return ERROR_CODE;
00711
00712
        memset (&srv_addr, 0, sizeof (srv_addr));
srv_addr.sin_family = AF_INET;
srv_addr.sin_port = htons (p);
srv_addr.sin_addr.s_addr = hton1 (INADDR_ANY);
00713
00714
00715
00716
00717
00718
        if (bind (sock, (const struct sockaddr *)&srv_addr, sizeof (srv_addr)) < 0)</pre>
00719
00720
            return ERROR CODE;
00721
00722
00723
        if (listen (sock, ACCEPT_QUEUE) < 0)</pre>
00724
00725
            return ERROR_CODE;
00726
          }
00727
00728
        return sock;
00729 }
00730
00731 [[nodiscard]]
00732 struct saurion *
00733 saurion_create (uint32_t n_threads)
00734 {
00735
        LOG_INIT (" ");
00736
        struct saurion *p = (struct saurion *)malloc (sizeof (struct saurion));
00737
        if (!p)
        {
00738
            LOG_END (" ");
00739
00740
            return NULL;
00741
00742
        int ret = 0;
00743
        ret = pthread_mutex_init (&p->status_m, NULL);
00744
        if (ret)
00745
            free (p);
LOG_END (" ");
00746
00747
00748
            return NULL;
00749
00750
        ret = pthread_cond_init (&p->status_c, NULL);
00751
        if (ret)
00752
            free (p);
LOG_END (" ");
00753
00754
00755
            return NULL;
00756
00757
        p->m_rings
00758
            = (pthread mutex t *) malloc (n threads * sizeof (pthread mutex t));
00759
        if (!p->m_rings)
00760
         {
            free (p);
LOG_END (" ");
00761
00762
00763
            return NULL;
00764
00765
        for (uint32_t i = 0; i < n_threads; ++i)</pre>
```

7.14 low\_saurion.c 91

```
00767
            pthread_mutex_init (&(p->m_rings[i]), NULL);
00768
00769
        p->ss = 0;
        n_threads = (n_threads < 2 ? 2 : n_threads);
n_threads = (n_threads > NUM_CORES ? NUM_CORES : n_threads);
00770
00771
00772
        p->n_threads = n_threads;
00773
        p->status = 0;
00774
        p->list = NULL;
        p->cb.on_connected = NULL;
00775
00776
        p->cb.on_connected_arg = NULL;
        p->cb.on_readed = NULL;
00777
00778
        p->cb.on_readed_arg = NULL;
00779
        p->cb.on_wrote = NULL;
00780
        p->cb.on_wrote_arg = NULL;
00781
        p->cb.on_closed = NULL;
00782
        p->cb.on_closed_arg = NULL;
00783
        p->cb.on_error = NULL;
00784
        p->cb.on_error_arg = NULL;
        p->next = 0;
p->efds = (int *)malloc (sizeof (int) * p->n_threads);
00785
00786
00787
        if (!p->efds)
00788
          {
00789
            free (p->m_rings);
            free (p);
LOG_END (" ");
00790
00791
00792
             return NULL;
00793
        for (uint32_t i = 0; i < p->n_threads; ++i)
00794
00795
            p->efds[i] = eventfd (0, EFD_NONBLOCK);
00796
00797
             if (p->efds[i] == ERROR_CODE)
00798
00799
                 for (uint32_t j = 0; j < i; ++j)
                   close (p->efds[j]);
}
00800
00801
00802
00803
                 free (p->efds);
00804
                 free (p->m_rings);
                free (p);
LOG_END (" ");
00805
00806
                 return NULL;
00807
00808
00809
        p->rings
00810
00811
            = (struct io_uring *) malloc (sizeof (struct io_uring) * p->n_threads);
00812
        if (!p->rings)
00813
00814
            for (uint32_t j = 0; j < p->n_threads; ++j)
00815
00816
                close (p->efds[j]);
00817
00818
             free (p->efds);
00819
             free (p->m_rings);
            free (p);
LOG_END (" ");
00820
00821
            return NULL;
00822
00823
00824
        for (uint32_t i = 0; i < p->n_threads; ++i)
00825
00826
            memset (&p->rings[i], 0, sizeof (struct io_uring));
             ret = io_uring_queue_init (SAURION_RING_SIZE, &p->rings[i], 0);
00827
00828
             if (ret)
00829
00830
                 for (uint32_t j = 0; j < p->n_threads; ++j)
                  close (p->efds[j]);
}
00831
00832
00833
00834
                 free (p->efds);
00835
                 free (p->rings);
00836
                 free (p->m_rings);
                free (p);
LOG_END (" ");
00837
00838
00839
                 return NULL;
00840
               }
00841
        p->pool = threadpool_create (p->n_threads);
LOG_END (" ");
00842
00843
00844
        return p;
00845 }
00846
00847 [[nodiscard]]
00848 static int
00849 saurion_worker_master_loop_it (struct saurion \starconst s,
00850
                                        struct sockaddr_in *client_addr,
00851
                                        socklen_t *client_addr_len)
00852 {
```

```
LOG_INIT (" ");
00854
        struct io_uring ring = s->rings[0];
00855
        struct io_uring_cqe *cqe = NULL;
        int ret = io_uring_wait_cqe (&ring, &cqe);
if (ret < 0)</pre>
00856
00857
00858
          {
             free (cqe);
LOG_END (" ");
00859
00860
             return CRITICAL_CODE;
00861
00862
00863
        struct request *req = (struct request *)cqe->user_data;
00864
        if (!rea)
00865
          {
             io_uring_cqe_seen (&s->rings[0], cqe);
LOG_END (" ");
00866
00867
             return SUCCESS_CODE;
00868
00869
00870
        if (cge->res < 0)
00871
             list_delete_node (&s->list, req);
LOG_END (" ");
00872
00873
             return CRITICAL_CODE;
00874
00875
00876
        if (req->client_socket == s->efds[0])
00877
          {
00878
             io_uring_cqe_seen (&s->rings[0], cqe);
             list_delete_node (&s->list, req);
LOG_END (" ");
00879
00880
00881
             return ERROR_CODE;
00882
00883
        io_uring_cqe_seen (&s->rings[0], cqe);
        switch (req->event_type)
00884
00885
00886
           case EV_ACC:
            handle_accept (s, cqe->res);
add_accept (s, client_addr, client_addr_len);
00887
00888
00889
             add_read (s, cqe->res);
             list_delete_node (&s->list, req);
00891
             break;
00892
          case EV_REA:
00893
             if (cqe->res < 0)</pre>
00894
              {
00895
                 handle error (s, reg);
00896
00897
             if (cqe->res < 1)
00898
               {
00899
                 handle_close (s, req);
00900
00901
             if (cge->res > 0)
00902
00903
                 handle_read (s, req);
00904
00905
             list_delete_node (&s->list, req);
          break;
case EV_WRI:
00906
00907
            handle_write (s, req->client_socket);
list_delete_node (&s->list, req);
00908
00909
00910
             break;
00911
        LOG_END (" ");
00912
        return SUCCESS_CODE;
00913
00914 }
00915
00916 void
00917 saurion_worker_master (void *arg)
00918 {
        LOG_INIT (" ");
00919
        struct saurion *const s = (struct saurion *)arg;
00920
        struct sockaddr_in client_addr;
00921
00922
        socklen_t client_addr_len = sizeof (client_addr);
00923
00924
        add_efd (s, s->efds[0], 0);
00925
        add_accept (s, &client_addr, &client_addr_len);
00926
00927
        pthread mutex lock (&s->status m);
00928
         ++s->status;
00929
        pthread_cond_broadcast (&s->status_c);
00930
        pthread_mutex_unlock (&s->status_m);
00931
         while (1)
00932
00933
00934
                  = saurion_worker_master_loop_it (s, &client_addr, &client_addr_len);
00935
             if (ret == ERROR_CODE || ret == CRITICAL_CODE)
00936
00937
                 break;
               }
00938
00939
           }
```

7.14 low\_saurion.c 93

```
pthread_mutex_lock (&s->status_m);
00941
          -s->status;
00942
        pthread_cond_signal (&s->status_c);
00943
        pthread_mutex_unlock (&s->status_m);
00944
        LOG END (" ");
00945
        return:
00946 }
00947
00948 [[nodiscard]]
00949 static int
00950 saurion_worker_slave_loop_it (struct saurion *const s, const int sel)
00951 {
00952
        LOG_INIT (" ");
00953
        struct io_uring ring = s->rings[sel];
00954
        struct io_uring_cqe *cqe = NULL;
00955
00956
        add_efd (s, s->efds[sel], sel);
        int ret = io_uring_wait_cqe (&ring, &cqe);
00957
        if (ret < 0)
00958
00959
          {
            free (cqe);
LOG_END (" ");
00960
00961
            return CRITICAL_CODE;
00962
00963
00964
        struct request *req = (struct request *)cqe->user_data;
00965
        if (!req)
00966
            io_uring_cqe_seen (&ring, cqe);
LOG_END (" ");
return SUCCESS_CODE;
00967
00968
00969
00970
00971
        if (cqe->res < 0)
00972
            list_delete_node (&s->list, req);
LOG_END (" ");
00973
00974
            return CRITICAL_CODE;
00975
00976
00977
        if (req->client_socket == s->efds[sel])
00978
          {
00979
            io_uring_cqe_seen (&ring, cqe);
00980
            list_delete_node (&s->list, req);
LOG_END (" ");
00981
            return ERROR_CODE;
00982
00983
00984
        io_uring_cqe_seen (&ring, cqe);
00985
        switch (req->event_type)
00986
00987
          case EV_REA:
00988
            if (cqe->res < 0)</pre>
00989
              {
00990
                handle_error (s, req);
00991
00992
            if (cqe->res < 1)
00993
00994
                handle_close (s, reg);
00995
00996
            if (cqe->res > 0)
00997
              {
00998
                handle_read (s, req);
00999
01000
            list_delete_node (&s->list, req);
01001
            break;
01002
          case EV_WRI:
01003
           handle_write (s, req->client_socket);
01004
             list_delete_node (&s->list, req);
01005
            break;
01006
        LOG_END (" ");
01007
01008
        return SUCCESS_CODE;
01009 }
01010
01011 void
01012 saurion_worker_slave (void *arg)
01013 {
        LOG_INIT (" ");
01014
01015
        struct saurion_wrapper *const ss = (struct saurion_wrapper *)arg;
01016
        struct saurion *s = ss->s;
        const int sel = ss->sel;
01017
01018
        free (ss);
01019
01020
        add efd (s, s->efds[sel], sel);
01021
01022
        pthread_mutex_lock (&s->status_m);
01023
         ++s->status;
01024
        pthread_cond_broadcast (&s->status_c);
01025
        pthread_mutex_unlock (&s->status_m);
01026
        while (1)
```

```
{
01028
            int res = saurion_worker_slave_loop_it (s, sel);
01029
            if (res == ERROR_CODE || res == CRITICAL_CODE)
01030
             {
01031
                break:
              }
01032
01033
01034
        pthread_mutex_lock (&s->status_m);
01035
         --s->status;
01036
        pthread_cond_signal (&s->status_c);
01037
        pthread_mutex_unlock (&s->status_m);
        LOG_END (" ");
01038
01039
        return;
01040 }
01041
01042 [[nodiscard]]
01043 int
01044 saurion start (struct saurion *const s)
01045 {
01046
       pthread_mutex_init (&print_mutex, NULL);
01047
        threadpool_init (s->pool);
01048
        threadpool_add (s->pool, saurion_worker_master, s);
        struct saurion_wrapper *ss = NULL;
01049
01050
        for (uint32_t i = 1; i < s->n_threads; ++i)
01051
01052
           ss = (struct saurion_wrapper *)malloc (sizeof (struct saurion_wrapper));
01053
            if (!ss)
01054
             {
01055
                return ERROR_CODE;
              }
01056
            ss->s = s;
ss->sel = i;
01057
01058
01059
            threadpool_add (s->pool, saurion_worker_slave, ss);
01060
        pthread_mutex_lock (&s->status_m);
while (s->status < (int)s->n_threads)
01061
01062
01063
        {
01064
           pthread_cond_wait (&s->status_c, &s->status_m);
01065
01066
        pthread_mutex_unlock (&s->status_m);
01067
        return SUCCESS_CODE;
01068 }
01069
01070 void
01071 saurion_stop (const struct saurion *const s)
01072 {
01073
        uint64 t u = 1;
        for (uint32_t i = 0; i < s->n_threads; ++i)
01074
01075
01076
            while (write (s->efds[i], &u, sizeof (u)) < 0)
01078
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
01079
01080
        threadpool_wait_empty (s->pool);
01081
01082 }
01084 void
01085 saurion_destroy (struct saurion *const s)
01086 {
01087
        pthread mutex lock (&s->status m);
01088
        while (s->status > 0)
01089
01090
            pthread_cond_wait (&s->status_c, &s->status_m);
01091
01092
        pthread_mutex_unlock (&s->status_m);
01093
        threadpool_destroy (s->pool);
        for (uint32_t i = 0; i < s->n_threads; ++i)
01094
01095
01096
            io_uring_queue_exit (&s->rings[i]);
01097
            pthread_mutex_destroy (&s->m_rings[i]);
01098
01099
        free (s->m_rings);
        list free (&s->list);
01100
        for (uint32_t i = 0; i < s->n_threads; ++i)
01101
01102
01103
            close (s->efds[i]);
01104
01105
        free (s->efds);
        if (!s->ss)
01106
01107
         {
01108
            close (s->ss);
01109
01110
        free (s->rings);
01111
        pthread_mutex_destroy (&s->status_m);
01112
        pthread_cond_destroy (&s->status_c);
01113
        free (s):
```

```
01114 }
01115
01116 void
01117 saurion_send (struct saurion *const s, const int fd, const char *const msg)
01118 {
01119 add_write (s, fd, msg, next (s));
01120 }
```

# 7.15 / w/saurion/saurion/src/main.c File Reference

```
#include <pthread.h>
#include <stdio.h>
Include dependency graph for main.c:
```

#### 7.16 main.c

```
Go to the documentation of this file.
00001 #include <pthread.h> // for pthread_create, pthread_join, pthread_t 00002 #include <stdio.h> // for printf, fprintf, NULL, stderr
00003
00004 int counter = 0;
00005
00006 void \star
00007 increment (void *arg)
00008 {
00009
        int id = *((int *)arg);
        for (int i = 0; i < 100000; ++i)
00010
00012
             counter++;
00013
            if (i % 10000 == 0)
00014
              {
                printf ("Thread %d at iteration %d\n", id, i);
00015
00016
00017
00018
       printf ("Thread %d finished\n", id);
00019
        return NULL;
00020 }
00021
00022 int
00023 main ()
00024 {
00025
        pthread_t t1;
00026
        pthread_t t2;
00027
        int id1 = 1;
00028
        int id2 = 2;
00029
00030
        printf ("Starting threads...\n");
00031
00032
        if (pthread_create (&t1, NULL, increment, &id1))
00033
         {
            fprintf (stderr, "Error creating thread 1\n");
00034
00035
            return 1;
00036
00037
        if (pthread_create (&t2, NULL, increment, &id2))
00038
            fprintf (stderr, "Error creating thread 2\n");
00039
00040
            return 1;
00041
00042
00043
        printf ("Waiting for thread 1 to join...\n");
00044
         if (pthread_join (t1, NULL))
00045
             fprintf (stderr, "Error joining thread 1\n");
00046
00047
            return 2;
00048
00049
        printf ("Thread 1 joined\n");
00050
        printf ("Waiting for thread 2 to join...\n");
00051
00052
        if (pthread_join (t2, NULL))
00053
00054
             fprintf (stderr, "Error joining thread 2\n");
00055
            return 2;
00056
00057
        printf ("Thread 2 joined\n");
00058
00059
        printf ("Final counter value: %d\n", counter);
00060
        return 0;
00061 }
```

# 7.17 / w/saurion/saurion/src/saurion.cpp File Reference

```
#include "saurion.hpp"
#include "low_saurion.h"
Include dependency graph for saurion.cpp:
```

# 7.18 saurion.cpp

#### Go to the documentation of this file.

```
00001 #include "saurion.hpp"
00002
00003 #include "low_saurion.h" // for saurion, saurion_create, saurion_destroy
00004
00005 Saurion::Saurion (const uint32_t thds, const int sck) noexcept
00006 {
        this->s = saurion_create (thds);
00007
00008
       if (!this->s)
00009
00010
            return:
00011
00012
       this->s->ss = sck;
00013 }
00014
00015 Saurion::~Saurion () { saurion_destroy (this->s); }
00016
00017 void
00018 Saurion::init () noexcept
00019 {
00020
       if (!saurion_start (this->s))
00021
00022
            return:
00023
00024 }
00025
00026 void
00027 Saurion::stop () const noexcept
00028 {
00029
        saurion_stop (this->s);
00030 }
00032 Saurion *
00033 Saurion::on_connected (Saurion::ConnectedCb ncb, void *arg) noexcept
00034 {
       s->cb.on_connected = ncb;
00035
00036
       s->cb.on_connected_arg = arg;
00037
       return this;
00038 }
00039
00040 Saurion *
00041 Saurion::on_readed (Saurion::ReadedCb ncb, void *arg) noexcept
00042 {
00043
       s->cb.on_readed = ncb;
00044 s->cb.on_readed_arg = arg;
00045
       return this;
00046 }
00047
00048 Saurion *
00049 Saurion::on_wrote (Saurion::WroteCb ncb, void *arg) noexcept
00050 {
00051 s->cb.on_wrote = ncb;
00052 s->cb.on_wrote_arg = arg;
00053
       return this;
00054 }
00055
00056 Saurion *
00057 Saurion::on_closed (Saurion::ClosedCb ncb, void *arg) noexcept
00058 {
00059
       s->cb.on_closed = ncb;
00060 s->cb.on_closed_arg = arg;
00061
       return this;
00062 }
00063
00064 Saurion *
00065 Saurion::on_error (Saurion::ErrorCb ncb, void *arg) noexcept
00066 {
00067
       s->cb.on error = ncb;
00068 s->cb.on_error_arg = arg;
00069
       return this;
```

```
00070 }
00071
00072 void
00073 Saurion::send (const int fd, const char *const msg) noexcept
00074 {
00075    saurion_send (this->s, fd, msg);
00076 }
```

# 7.19 /\_w/saurion/saurion/src/threadpool.c File Reference

```
#include "threadpool.h"
#include "config.h"
#include <nanologger.h>
#include <pthread.h>
#include <stdio.h>
#include <stdib.h>
```

Include dependency graph for threadpool.c:

#### **Classes**

- struct task
- · struct threadpool

#### **Macros**

- #define TRUE 1
- #define FALSE 0

#### **Functions**

- struct threadpool \* threadpool\_create (size\_t num\_threads)
- struct threadpool \* threadpool\_create\_default (void)
- void \* threadpool\_worker (void \*arg)
- void threadpool\_init (struct threadpool \*pool)
- void threadpool\_add (struct threadpool \*pool, void(\*function)(void \*), void \*argument)
- void threadpool\_stop (struct threadpool \*pool)
- int threadpool\_empty (struct threadpool \*pool)
- void threadpool\_wait\_empty (struct threadpool \*pool)
- void threadpool\_destroy (struct threadpool \*pool)

#### 7.19.1 Macro Definition Documentation

#### 7.19.1.1 FALSE

```
#define FALSE 0
```

Definition at line 9 of file threadpool.c.

#### 7.19.1.2 TRUE

```
#define TRUE 1
```

Definition at line 8 of file threadpool.c.

#### 7.19.2 Function Documentation

#### 7.19.2.1 threadpool\_worker()

```
Definition at line 107 of file threadpool.c.
```

```
00108 {
        LOG_INIT (" ");
00110
        struct threadpool *pool = (struct threadpool *)arg;
00111
        while (TRUE)
00112
            pthread_mutex_lock (&pool->queue_lock);
while (pool->task_queue_head == NULL && !pool->stop)
00113
00114
00115
              {
00116
                pthread_cond_wait (&pool->queue_cond, &pool->queue_lock);
00117
00118
00119
            if (pool->stop && pool->task_queue_head == NULL)
00120
             {
00121
                pthread_mutex_unlock (&pool->queue_lock);
00122
                break;
00123
00124
00125
            struct task *task = pool->task_queue_head;
00126
            if (task != NULL)
00127
              {
00128
                pool->task_queue_head = task->next;
00129
                 if (pool->task_queue_head == NULL)
00130
                  pool->task_queue_tail = NULL;
00131
00132
                 if (pool->task_queue_head == NULL)
00133
00134
                     pthread_cond_signal (&pool->empty_cond);
00135
00136
00137
            pthread_mutex_unlock (&pool->queue_lock);
00138
00139
            if (task != NULL)
00140
00141
                 task->function (task->argument);
00142
                 free (task);
00143
00144
       LOG_END (" ");
pthread_exit (NULL);
00145
00146
00147
        return NULL;
00148 }
```

# 7.20 threadpool.c

#### Go to the documentation of this file.

```
00001 #include "threadpool.h"
00002 #include "config.h" // for NUM_CORES
00003 #include <nanologger.h> // for LOG_END, LOG_INIT
00004 #include <pthread.h> // for pthread_mutex_unlock, pthread_mutex_lock
00005 #include <stdio.h> // for perror
00006 #include <stdib.h> // for free, malloc
```

7.20 threadpool.c 99

```
00007
00008 #define TRUE 1
00009 #define FALSE 0
00010
00011 struct task
00012 {
        void (*function) (void *);
00013
00014
        void *argument;
00015
       struct task *next;
00016 };
00017
00018 struct threadpool
00019 {
00020
        pthread_t *threads;
00021
        size_t num_threads;
00022
        struct task *task_queue_head;
00023
        struct task *task_queue_tail;
00024
        pthread_mutex_t queue_lock;
        pthread_cond_t queue_cond;
00026
        pthread_cond_t empty_cond;
00027
        int stop;
00028
        int started;
00029 };
00030
00031 struct threadpool *
00032 threadpool_create (size_t num_threads)
00033 {
        LOG_INIT (" ");
00034
00035
        struct threadpool *pool = malloc (sizeof (struct threadpool));
        if (pool == NULL)
00036
00037
00038
            perror ("Failed to allocate threadpool");
00039
             LOG_END (" ");
00040
             return NULL;
00041
        if (num_threads < 3)</pre>
00042
00043
        {
00044
            num_threads = 3;
00045
00046
        if (num_threads > NUM_CORES)
00047
00048
            num threads = NUM CORES;
00049
00050
00051
        pool->num_threads = num_threads;
00052
        pool->threads = malloc (sizeof (pthread_t) * num_threads);
00053
         if (pool->threads == NULL)
00054
00055
            perror ("Failed to allocate threads array");
            free (pool);
LOG_END (" ");
00056
00057
00058
            return NULL;
00059
00060
00061
        pool->task_queue_head = NULL;
        pool->task_queue_tail = NULL;
pool->stop = FALSE;
00062
00063
00064
        pool->started = FALSE;
00065
00066
        if (pthread_mutex_init (&pool->queue_lock, NULL) != 0)
00067
         {
             perror ("Failed to initialize mutex");
00068
00069
             free (pool->threads);
00070
            free (pool);
LOG_END (" ");
00071
00072
             return NULL;
00073
00074
        if (pthread_cond_init (&pool->queue_cond, NULL) != 0)
00075
00076
00077
            perror ("Failed to initialize condition variable");
00078
             pthread_mutex_destroy (&pool->queue_lock);
00079
             free (pool->threads);
08000
            free (pool);
LOG_END (" ");
00081
00082
            return NULL;
00083
00084
00085
        if (pthread_cond_init (&pool->empty_cond, NULL) != 0)
00086
00087
            perror ("Failed to initialize empty condition variable");
            pthread_mutex_destroy (&pool->queue_lock);
pthread_cond_destroy (&pool->queue_cond);
00088
00089
00090
             free (pool->threads);
            free (pool);
LOG_END (" ");
00091
00092
             return NULL;
00093
```

```
00094
         }
00095
       LOG_END (" ");
00096
00097
       return pool;
00098 }
00099
00100 struct threadpool *
00101 threadpool_create_default (void)
00102 {
00103
        return threadpool_create (NUM_CORES);
00104 }
00105
00106 void *
00107 threadpool_worker (void *arg)
00108 {
00109
       LOG_INIT (" ");
        struct threadpool *pool = (struct threadpool *)arg;
00110
        while (TRUE)
00111
00112
00113
            pthread_mutex_lock (&pool->queue_lock);
00114
            while (pool->task_queue_head == NULL && !pool->stop)
00115
00116
                pthread_cond_wait (&pool->queue_cond, &pool->queue_lock);
00117
00118
00119
            if (pool->stop && pool->task_queue_head == NULL)
00120
00121
               pthread_mutex_unlock (&pool->queue_lock);
00122
                break;
00123
             }
00124
00125
            struct task *task = pool->task_queue_head;
00126
            if (task != NULL)
00127
00128
               pool->task_queue_head = task->next;
                if (pool->task_queue_head == NULL)
00129
                pool->task_queue_tail = NULL;
00130
00131
00132
                if (pool->task_queue_head == NULL)
00133
00134
                   pthread_cond_signal (&pool->empty_cond);
                 }
00135
00136
00137
           pthread_mutex_unlock (&pool->queue_lock);
00138
00139
            if (task != NULL)
00140
             {
               task->function (task->argument);
00141
00142
               free (task);
00143
             }
00144
00145
       LOG_END (" ");
00146
       pthread_exit (NULL);
00147
        return NULL;
00148 }
00149
00150 void
00151 threadpool_init (struct threadpool *pool)
00152 {
        LOG_INIT (" ");
00153
        if (pool == NULL || pool->started)
00154
00155
00156
            LOG_END (" ");
00157
           return;
00158
00159
        for (size_t i = 0; i < pool->num_threads; i++)
00160
            if (pthread_create (&pool->threads[i], NULL, threadpool_worker,
00161
00162
                                (void *)pool)
00163
               != 0)
00164
               perror ("Failed to create thread");
00165
               pool->stop = TRUE;
00166
00167
                break:
00168
             }
00169
00170
        pool->started = TRUE;
00171
       LOG_END (" ");
00172 }
00173
00174 void
00175 threadpool_add (struct threadpool *pool, void (*function) (void *),
                      void *argument)
00176
00177 {
00178
       LOG_INIT (" ");
        if (pool == NULL || function == NULL)
00179
00180
```

7.20 threadpool.c 101

```
LOG_END (" ");
00181
00182
           return;
00183
00184
00185
        struct task *new task = malloc (sizeof (struct task));
00186
        if (new_task == NULL)
00187
00188
            perror ("Failed to allocate task");
00189
            LOG_END (" ");
00190
            return;
          }
00191
00192
        new_task->function = function;
new_task->argument = argument;
00193
00194
        new_task->next = NULL;
00195
00196
00197
        pthread_mutex_lock (&pool->queue_lock);
00198
00199
        if (pool->task_queue_head == NULL)
00200
         {
00201
            pool->task_queue_head = new_task;
            pool->task_queue_tail = new_task;
00202
00203
00204
        else
00205
         {
00206
           pool->task_queue_tail->next = new_task;
00207
            pool->task_queue_tail = new_task;
00208
00209
        pthread_cond_signal (&pool->queue_cond);
00210
00211
        pthread_mutex_unlock (&pool->queue_lock);
00212
        LOG_END (" ");
00213 }
00214
00215 void
00216 threadpool_stop (struct threadpool *pool)
00217 {
        LOG_INIT (" ");
00219
        if (pool == NULL || !pool->started)
00220
            LOG_END (" ");
00221
00222
            return;
00223
00224
        threadpool_wait_empty (pool);
00225
00226
        pthread_mutex_lock (&pool->queue_lock);
00227
        pool->stop = TRUE;
00228
        pthread_cond_broadcast (&pool->queue_cond);
00229
        pthread_mutex_unlock (&pool->queue_lock);
00230
00231
        for (size_t i = 0; i < pool->num_threads; i++)
00232
00233
           pthread_join (pool->threads[i], NULL);
00234
        pool->started = FALSE;
00235
00236
        LOG_END (" ");
00237 }
00238
00239 int
00240 threadpool_empty (struct threadpool *pool)
00241 {
        LOG_INIT (" ");
00242
00243
        if (pool == NULL)
00244
            LOG_END (" ");
00245
00246
            return TRUE;
00247
00248
        pthread_mutex_lock (&pool->queue_lock);
        int empty = (pool->task_queue_head == NULL);
00249
        pthread_mutex_unlock (&pool->queue_lock);
00251
        LOG_END (" ");
00252
        return empty;
00253 }
00254
00255 void
00256 threadpool_wait_empty (struct threadpool *pool)
00257 {
00258
        LOG_INIT (" ");
00259
        if (pool == NULL)
00260
         {
            LOG_END (" ");
00261
00262
            return;
00263
00264
        pthread_mutex_lock (&pool->queue_lock);
00265
        while (pool->task_queue_head != NULL)
00266
00267
            pthread cond wait (&pool->empty cond, &pool->gueue lock);
```

```
00268
          pthread_mutex_unlock (&pool->queue_lock);
LOG_END (" ");
00269
00270
00271 }
00272
00273 void
00274 threadpool_destroy (struct threadpool *pool)
00275 {
          LOG_INIT (" ");
if (pool == NULL)
{
00276
00277
00278
00279
               LOG_END (" ");
00280
               return;
00281
00282
          threadpool_stop (pool);
00283
          pthread_mutex_lock (&pool->queue_lock);
struct task *task = pool->task_queue_head;
while (task != NULL)
00284
00285
00286
00287
           {
               struct task *tmp = task;
task = task->next;
00288
00289
              free (tmp);
00290
00291
          pthread_mutex_unlock (&pool->queue_lock);
00292
00293
00294
          pthread_mutex_destroy (&pool->queue_lock);
          pthread_cond_destroy (&pool->queue_cond);
pthread_cond_destroy (&pool->empty_cond);
00295
00296
00297
         free (pool->threads);
free (pool);
LOG_END (" ");
00298
00299
00300
00301 }
```

# Index

| /_w/saurion/saurion/include/linked_list.h, 53, 55   | EV_ERR               |
|---|----------------------|
| /w/saurion/saurion/include/low_saurion.h, 55, 61    | low_saurion.c, 72    |
| /w/saurion/saurion/include/low_saurion_secret.h, 62 | EV_REA               |
| /_w/saurion/saurion/include/saurion.hpp, 63         | low_saurion.c, 72    |
| /w/saurion/saurion/include/threadpool.h, 64         | EV_WAI               |
| /w/saurion/saurion/src/linked_list.c, 65, 68        | low_saurion.c, 72    |
| /w/saurion/saurion/src/low_saurion.c, 70, 82        | EV_WRI               |
| /w/saurion/saurion/src/main.c, 95                   | low_saurion.c, 72    |
| /w/saurion/saurion/src/saurion.cpp, 96              | event_type           |
| /_w/saurion/saurion/src/threadpool.c, 97, 98        | request, 32          |
| POSIX_C_SOURCE                                      | EXTERNAL_set_socket  |
| LowSaurion, 12                                      | LowSaurion, 13       |
| $\sim$ Saurion                                      | •                    |
| Saurion, 39   | FALSE                |
| ,   | threadpool.c, 97     |
| add_accept  | free_node            |
| low_saurion.c, 73                                   | linked_list.c, 66    |
| add efd   | free_request         |
| low saurion.c, 73                                   | LowSaurion, 13       |
| add read  | function             |
| low_saurion.c, 74                                   | task, 50             |
| add_read_continue                                   | •                    |
| low_saurion.c, 75                                   | handle_accept        |
| add_write   | low_saurion.c, 76    |
| low_saurion.c, 75                                   | handle_close         |
| allocate_iovec                                      | low_saurion.c, 76    |
| LowSaurion, 12                                      | handle_error         |
| argument  | low_saurion.c, 76    |
| task, 50  | handle_read          |
| tack, oo  | low_saurion.c, 77    |
| children  | handle_write         |
| Node, 31  | low_saurion.c, 77    |
| client_socket                                       | htonll               |
| request, 32   | low_saurion.c, 78    |
| ClosedCb  | 10W_5aa11011.5, 76   |
| Saurion, 38   | init                 |
| ConnectedCb   | Saurion, 40          |
| Saurion, 38   | initialize iovec     |
| create_node   | LowSaurion, 14       |
| linked_list.c, 65                                   | iov                  |
| III IKEU_IISt.C, 03                                 | request, 33          |
| efds  | iovec count          |
| low_saurion.h, 56                                   | request, 33          |
| saurion, 35   | 1044001, 00          |
| empty_cond  | linked list.c        |
| threadpool, 51                                      | create_node, 65      |
| ErrorCb   | free_node, 66        |
| Saurion, 38   | list_delete_node, 66 |
| •   | list_free, 67        |
| EV_ACC  | list insert, 67      |
| low_saurion.c, 72                                   | 113C11, 07           |

| list_mutex, 68                    | on_readed, 59           |
|-----------------------------------|-------------------------|
| linked_list.h                     | on_readed_arg, 59       |
| list_delete_node, 53              | on_wrote, 59            |
| list_free, 54                     | on_wrote_arg, 60        |
| list_insert, 54                   | pool, 60                |
| list                              | rings, 60               |
| low_saurion.h, 56                 | ss, 60                  |
| saurion, 35                       | status, 60              |
| list delete node                  | status c, 61            |
| linked list.c, 66                 | status m, 61            |
| linked list.h, 53                 | LowSaurion, 9           |
| list free                         | _POSIX_C_SOURCE, 12     |
| linked list.c, 67                 | allocate_iovec, 12      |
| linked_list.h, 54                 | EXTERNAL_set_socket, 13 |
| list insert                       | free_request, 13        |
| linked list.c, 67                 | initialize_iovec, 14    |
| linked list.h, 54                 | PACKING_SZ, 12          |
| list_mutex                        | read_chunk, 15          |
| linked_list.c, 68                 | saurion_create, 19      |
|                                   |                         |
| low_saurion.c                     | saurion_destroy, 21     |
| add_accept, 73                    | saurion_send, 22        |
| add_efd, 73                       | saurion_start, 22       |
| add_read, 74                      | saurion_stop, 23        |
| add_read_continue, 75             | set_request, 23         |
| add_write, 75                     | and after the           |
| EV_ACC, 72                        | m_rings                 |
| EV_ERR, 72                        | low_saurion.h, 56       |
| EV_REA, 72                        | saurion, 35             |
| EV_WAI, 72                        | MAX                     |
| EV_WRI, 72                        | low_saurion.c, 72       |
| handle_accept, 76                 | MIN                     |
| handle_close, 76                  | low_saurion.c, 73       |
| handle_error, 76                  |                         |
| handle_read, 77                   | n_threads               |
| handle_write, 77                  | low_saurion.h, 57       |
| htonll, 78                        | saurion, 35             |
| MAX, 72                           | next                    |
| MIN, 73                           | low_saurion.c, 78       |
| next, 78                          | low_saurion.h, 57       |
| ntohll, 78                        | Node, 31                |
| print_mutex, 81                   | saurion, 35             |
| saurion_worker_master, 78         | task, 50                |
| saurion_worker_master_loop_it, 79 | next_iov                |
| saurion worker slave, 80          | request, 33             |
| :                                 | next offset             |
| saurion_worker_slave_loop_it, 80  | request, 33             |
| TIMEOUT_RETRY_SPEC, 81            | Node, 31                |
| low_saurion.h                     | children, 31            |
| efds, 56                          | next, 31                |
| list, 56                          | ptr, 31                 |
| m_rings, 56                       | size, 32                |
| n_threads, 57                     | ntohll                  |
| next, 57                          | low_saurion.c, 78       |
| on_closed, 57                     | num_threads             |
| on_closed_arg, 57                 |                         |
| on_connected, 58                  | threadpool, 51          |
| on_connected_arg, 58              | on closed               |
| on_error, 58                      | low saurion.h, 57       |
| on_error_arg, 59                  | <del>-</del>            |
|                                   | Saurion, 40             |

| saurion::saurion_callbacks, 43 | Node, 31                         |
|--------------------------------|----------------------------------|
| saurion_callbacks, 46          |                                  |
| on_closed_arg                  | queue_cond                       |
| low_saurion.h, 57              | threadpool, 52                   |
| saurion::saurion_callbacks, 43 | queue_lock                       |
| saurion_callbacks, 47          | threadpool, 52                   |
| on_connected                   | road abunit                      |
| low_saurion.h, 58              | read_chunk                       |
| Saurion, 40                    | LowSaurion, 15                   |
| saurion::saurion_callbacks, 43 | ReadedCb                         |
| saurion_callbacks, 47          | Saurion, 38                      |
| on_connected_arg               | request, 32                      |
| low_saurion.h, 58              | client_socket, 32                |
| saurion::saurion_callbacks, 44 | event_type, 32                   |
| saurion_callbacks, 47          | iov, 33                          |
| on_error                       | iovec_count, 33                  |
| low_saurion.h, 58              | next_iov, 33                     |
| Saurion, 40                    | next_offset, 33                  |
| saurion::saurion_callbacks, 44 | prev, 33                         |
| saurion_callbacks, 47          | prev_remain, 33                  |
| on_error_arg                   | prev_size, 34                    |
| low_saurion.h, 59              | rings                            |
| saurion::saurion_callbacks, 44 | low_saurion.h, 60<br>saurion, 36 |
| saurion_callbacks, 48          | Sauriori, 36                     |
| on_readed                      | S                                |
| low_saurion.h, 59              | Saurion, 42                      |
| Saurion, 40                    | saurion_wrapper, 49              |
| saurion::saurion_callbacks, 44 | Saurion, 37                      |
| saurion_callbacks, 48          | $\sim$ Saurion, 39               |
| on_readed_arg                  | ClosedCb, 38                     |
| low_saurion.h, 59              | ConnectedCb, 38                  |
| saurion::saurion_callbacks, 45 | ErrorCb, 38                      |
| saurion_callbacks, 48          | init, 40                         |
| on_wrote                       | on_closed, 40                    |
| low_saurion.h, 59              | on connected, 40                 |
| Saurion, 41                    | on_error, 40                     |
| saurion::saurion_callbacks, 45 | on_readed, 40                    |
| saurion_callbacks, 48          | on_wrote, 41                     |
| on_wrote_arg                   | operator=, 41                    |
| low_saurion.h, 60              | ReadedCb, 38                     |
| saurion::saurion_callbacks, 45 | s, 42                            |
| saurion_callbacks, 49          | Saurion, 39                      |
| operator=                      | send, 41                         |
| Saurion, 41                    | stop, 41                         |
| PACKING SZ                     | WroteCb, 38                      |
| LowSaurion, 12                 | saurion, 34                      |
| pool                           | efds, 35                         |
| low_saurion.h, 60              | list, 35                         |
| saurion, 36                    | m_rings, 35                      |
| prev                           | n_threads, 35                    |
| request, 33                    | next, 35                         |
| prev_remain                    | pool, 36                         |
| request, 33                    | rings, <mark>36</mark>           |
| prev_size                      | ss, 36                           |
| request, 34                    | status, 36                       |
| print_mutex                    | status_c, 36                     |
| low_saurion.c, 81              | status_m, 37                     |
| ptr                            | saurion::saurion_callbacks, 42   |
|                                |                                  |

| on_closed, 43                 | status_c                      |
|-------------------------------|-------------------------------|
| on_closed_arg, 43             | low_saurion.h, 61             |
| on_connected, 43              | saurion, 36                   |
| on_connected_arg, 44          | status_m                      |
| on_error, 44                  | low_saurion.h, 61             |
| on_error_arg, 44              | saurion, 37                   |
| on_readed, 44                 | stop                          |
| on_readed_arg, 45             | Saurion, 41                   |
| on_wrote, 45                  | threadpool, 52                |
| on_wrote_arg, 45              |                               |
| saurion_callbacks, 46         | task, 50                      |
| on_closed, 46                 | argument, 50                  |
| on_closed_arg, 47             | function, 50                  |
| on_connected, 47              | next, 50                      |
| on_connected_arg, 47          | task_queue_head               |
| on_error, 47                  | threadpool, 52                |
| on_error_arg, 48              | task_queue_tail               |
| on_readed, 48                 | threadpool, 52                |
| on_readed_arg, 48             | ThreadPool, 25                |
| on wrote, 48                  | threadpool_add, 25            |
| on_wrote_arg, 49              | threadpool_create, 26         |
| saurion_create                | threadpool_create_default, 27 |
| LowSaurion, 19                | threadpool_destroy, 27        |
| saurion_destroy               | threadpool_empty, 28          |
| LowSaurion, 21                | threadpool_init, 28           |
| saurion send                  | threadpool_stop, 28           |
| LowSaurion, 22                | threadpool_wait_empty, 29     |
| saurion start                 | threadpool, 51                |
| LowSaurion, 22                | empty_cond, 51                |
| saurion_stop                  | num_threads, 51               |
| LowSaurion, 23                | queue_cond, 52                |
| saurion_worker_master         | queue_lock, 52                |
| low_saurion.c, 78             | started, 52                   |
| saurion_worker_master_loop_it | stop, 52                      |
| low_saurion.c, 79             | task_queue_head, 52           |
| saurion_worker_slave          | task_queue_tail, 52           |
| low_saurion.c, 80             | threads, 52                   |
| saurion_worker_slave_loop_it  | threadpool.c                  |
| low_saurion.c, 80             | FALSE, 97                     |
| saurion_wrapper, 49           | threadpool_worker, 98         |
| s, 49                         | TRUE, 97                      |
| sel, 49                       | threadpool_add                |
| sel                           | ThreadPool, 25                |
| saurion_wrapper, 49           | threadpool_create             |
| send                          | ThreadPool, 26                |
| Saurion, 41                   | threadpool_create_default     |
| set_request                   | ThreadPool, 27                |
| LowSaurion, 23                | threadpool_destroy            |
| size                          | ThreadPool, 27                |
| Node, 32                      | threadpool_empty              |
| SS                            | ThreadPool, 28                |
| low_saurion.h, 60             | threadpool_init               |
| saurion, 36                   | ThreadPool, 28                |
| started                       | threadpool_stop               |
| threadpool, 52                | ThreadPool, 28                |
| status                        | threadpool_wait_empty         |
| low_saurion.h, 60             | ThreadPool, 29                |
| saurion, 36                   | threadpool_worker             |
| ,                             | threadpool.c, 98              |
|                               |                               |

threads
threadpool, 52
TIMEOUT\_RETRY\_SPEC
low\_saurion.c, 81
TRUE
threadpool.c, 97
WroteCb
Saurion, 38