Saurion

Generated by Doxygen 1.9.4

Todo List		1
Module Index		3
2.1 Modules		3
Class Index		5
3.1 Class List		5
Eila Index		7
File Index 4.1 File List		7
4.1 File List		,
Module Documentation		9
5.1 LowSaurion		9
5.1.1 Detailed Description	1	10
5.1.2 Macro Definition Documentation	1	12
5.1.2.1 _POSIX_C_SOURCE	1	12
5.1.2.2 PACKING_SZ	1	12
5.1.3 Function Documentation	1	13
5.1.3.1 allocate_iovec()	1	13
5.1.3.2 free_request()	1	13
5.1.3.3 initialize_iovec()	1	14
5.1.3.4 read_chunk()		15
5.1.3.5 saurion_create()		17
5.1.3.6 saurion_destroy()		18
5.1.3.7 saurion_send()		19
5.1.3.8 saurion_set_socket()		19
5.1.3.9 saurion_start()		20
5.1.3.10 saurion_stop()		21
5.1.3.11 set_request()		21
5.2 ThreadPool		23
5.2.1 Detailed Description		23
5.2.2 Function Documentation		23
5.2.2.1 threadpool_add()		23
5.2.2.2 threadpool_create()		24
5.2.2.3 threadpool_create_default()		25
5.2.2.4 threadpool_destroy()		25
5.2.2.5 threadpool_empty()		26
5.2.2.6 threadpool_init()		26
5.2.2.7 threadpool_stop()		26
5.2.2.8 threadpool_wait_empty()		27
		20
Class Documentation 6.1 chunk parame Struct Peferance		29
6.1 chunk_params Struct Reference		29 20
6.1.1 Detailed Description		29

6.1.2 Member Data Documentation	29
6.1.2.1 cont_rem	29
6.1.2.2 cont_sz	30
6.1.2.3 curr_iov	30
6.1.2.4 curr_iov_off	30
6.1.2.5 dest	30
6.1.2.6 dest_off	30
6.1.2.7 dest_ptr	30
6.1.2.8 len	31
6.1.2.9 max_iov_cont	31
6.1.2.10 req	31
6.2 ClientInterface Class Reference	31
6.2.1 Detailed Description	32
6.2.2 Constructor & Destructor Documentation	32
6.2.2.1 ClientInterface() [1/3]	32
6.2.2.2 ~ClientInterface()	32
6.2.2.3 ClientInterface() [2/3]	32
6.2.2.4 ClientInterface() [3/3]	32
6.2.3 Member Function Documentation	32
6.2.3.1 clean()	33
6.2.3.2 connect()	33
6.2.3.3 disconnect()	33
6.2.3.4 getFifoPath()	33
6.2.3.5 getPort()	33
6.2.3.6 operator=() [1/2]	33
6.2.3.7 operator=() [2/2]	33
6.2.3.8 reads()	34
6.2.3.9 send()	34
6.2.4 Member Data Documentation	34
6.2.4.1 fifo	34
6.2.4.2 fifoname	34
6.2.4.3 pid	34
6.2.4.4 port	34
6.3 Node Struct Reference	35
6.3.1 Detailed Description	35
6.3.2 Member Data Documentation	35
6.3.2.1 children	35
6.3.2.2 next	35
6.3.2.3 ptr	35
6.3.2.4 size	36
6.4 request Struct Reference	36
6.4.1 Detailed Description	36

6.4.2 Member Data Documentation	36
6.4.2.1 client_socket	36
6.4.2.2 event_type	36
6.4.2.3 iov	37
6.4.2.4 iovec_count	37
6.4.2.5 next_iov	37
6.4.2.6 next_offset	37
6.4.2.7 prev	37
6.4.2.8 prev_remain	37
6.4.2.9 prev_size	38
6.5 saurion Struct Reference	38
6.5.1 Detailed Description	38
6.5.2 Member Data Documentation	38
6.5.2.1 cb	39
6.5.2.2 efds	39
6.5.2.3 list	39
6.5.2.4 m_rings	39
6.5.2.5 n_threads	39
6.5.2.6 next	40
6.5.2.7 pool	40
6.5.2.8 rings	40
6.5.2.9 ss	40
6.5.2.10 status	40
6.5.2.11 status_c	41
6.5.2.12 status_m	41
6.6 Saurion Class Reference	41
6.6.1 Detailed Description	42
6.6.2 Member Typedef Documentation	42
6.6.2.1 ClosedCb	42
6.6.2.2 ConnectedCb	42
6.6.2.3 ErrorCb	42
6.6.2.4 ReadedCb	42
6.6.2.5 WroteCb	43
6.6.3 Constructor & Destructor Documentation	43
6.6.3.1 Saurion() [1/3]	43
6.6.3.2 ~Saurion()	43
6.6.3.3 Saurion() [2/3]	43
6.6.3.4 Saurion() [3/3]	43
6.6.4 Member Function Documentation	44
6.6.4.1 init()	44
6.6.4.2 on_closed()	44
6.6.4.3 on_connected()	44

6.6.4.4 on_error()	 . 44
6.6.4.5 on_readed()	 . 45
6.6.4.6 on_wrote()	 . 45
6.6.4.7 operator=() [1/2]	 . 45
6.6.4.8 operator=() [2/2]	 . 45
6.6.4.9 send()	 . 45
6.6.4.10 stop()	 . 46
6.6.5 Member Data Documentation	 . 46
6.6.5.1 s	 . 46
6.7 saurion_callbacks Struct Reference	 . 46
6.7.1 Detailed Description	 . 47
6.7.2 Member Data Documentation	 . 47
6.7.2.1 on_closed	 . 47
6.7.2.2 on_closed_arg	 . 47
6.7.2.3 on_connected	 . 47
6.7.2.4 on_connected_arg	 . 48
6.7.2.5 on_error	 . 48
6.7.2.6 on_error_arg	 . 48
6.7.2.7 on_readed	 . 48
6.7.2.8 on_readed_arg	 . 49
6.7.2.9 on_wrote	 . 49
6.7.2.10 on_wrote_arg	 . 49
6.8 saurion_wrapper Struct Reference	 . 50
6.8.1 Detailed Description	 . 50
6.8.2 Member Data Documentation	 . 50
6.8.2.1 s	 . 50
6.8.2.2 sel	 . 50
6.9 task Struct Reference	 . 50
6.9.1 Detailed Description	 . 51
6.9.2 Member Data Documentation	 . 51
6.9.2.1 argument	 . 51
6.9.2.2 function	 . 51
6.9.2.3 next	 . 51
6.10 threadpool Struct Reference	 . 51
6.10.1 Detailed Description	 . 52
6.10.2 Member Data Documentation	 . 52
6.10.2.1 empty_cond	 . 52
6.10.2.2 num_threads	
6.10.2.3 queue_cond	 . 52
6.10.2.4 queue_lock	 . 52
6.10.2.5 started	 . 52
6.10.2.6 stop	 . 53

6.10.2.7 task_queue_head	53
6.10.2.8 task_queue_tail	53
6.10.2.9 threads	53
File Documentation	55
7.1 / w/saurion/saurion/include/client_interface.hpp File Reference	
7.1.1 Function Documentation	
7.1.1.1 set_fifoname()	
7.1.1.2 set_port()	
7.2 client_interface.hpp	
7.3 /w/saurion/saurion/include/linked_list.h File Reference	
7.3.1 Function Documentation	
7.3.1.1 list delete node()	
7.3.1.2 list_free()	
7.3.1.3 list_insert()	
7.4 linked_list.h	
7.5 / w/saurion/saurion/include/low_saurion.h File Reference	
7.5.1 Variable Documentation	60
7.5.1.1 cb	60
7.5.1.2 efds	60
7.5.1.3 list	60
7.5.1.4 m_rings	60
7.5.1.5 n_threads	60
7.5.1.6 next	61
7.5.1.7 on_closed	61
7.5.1.8 on_closed_arg	61
7.5.1.9 on_connected	61
7.5.1.10 on_connected_arg	62
7.5.1.11 on_error	62
7.5.1.12 on_error_arg	62
7.5.1.13 on_readed	62
7.5.1.14 on_readed_arg	63
7.5.1.15 on_wrote	63
7.5.1.16 on_wrote_arg	63
7.5.1.17 pool	64
7.5.1.18 rings	64
7.5.1.19 ss	64
7.5.1.20 status	64
7.5.1.21 status_c	64
7.5.1.22 status_m	65
7.6 low_saurion.h	65
7.7 /w/saurion/saurion/include/low_saurion_secret.h File Reference	66

7.8 low_saurion_secret.h
7.9 /w/saurion/saurion/include/saurion.hpp File Reference
7.10 saurion.hpp
7.11 /w/saurion/saurion/include/threadpool.h File Reference 6
7.12 threadpool.h
7.13 /w/saurion/src/linked_list.c File Reference
7.13.1 Function Documentation
7.13.1.1 create_node()
7.13.1.2 free_node()
7.13.1.3 list_delete_node()
7.13.1.4 list_free()
7.13.1.5 list_insert()
7.13.2 Variable Documentation
7.13.2.1 list_mutex
7.14 linked_list.c
7.15 /w/saurion/src/low_saurion.c File Reference
7.15.1 Macro Definition Documentation
7.15.1.1 EV_ACC
7.15.1.2 EV_ERR
7.15.1.3 EV_REA
7.15.1.4 EV_WAI
7.15.1.5 EV_WRI
7.15.1.6 MAX
7.15.1.7 MIN
7.15.2 Function Documentation
7.15.2.1 add_accept()
7.15.2.2 add_efd()
7.15.2.3 add_fd()
7.15.2.4 add_read()
7.15.2.5 add_read_continue()
7.15.2.6 add_write()
7.15.2.7 calculate_max_iov_content()
7.15.2.8 copy_data()
7.15.2.9 handle_accept()
7.15.2.10 handle_close()
7.15.2.11 handle_error()
7.15.2.12 handle_event_read()
7.15.2.13 handle_new_message()
7.15.2.14 handle_partial_message()
7.15.2.15 handle_previous_message()
7.15.2.16 handle_read()
7.15.2.17 handle_write()

113

7.15.2.18 htonll()	84
7.15.2.19 next()	85
7.15.2.20 ntohll()	85
7.15.2.21 prepare_destination()	85
7.15.2.22 read_chunk_free()	86
7.15.2.23 saurion_worker_master()	86
7.15.2.24 saurion_worker_master_loop_it()	87
7.15.2.25 saurion_worker_slave()	87
7.15.2.26 saurion_worker_slave_loop_it()	88
7.15.2.27 validate_and_update()	89
7.15.3 Variable Documentation	89
7.15.3.1 TIMEOUT_RETRY_SPEC	89
7.16 low_saurion.c	89
7.17 /w/saurion/src/main.c File Reference	104
7.18 main.c	104
7.19 /w/saurion/saurion/src/saurion.cpp File Reference	104
7.20 saurion.cpp	105
7.21 /w/saurion/saurion/src/threadpool.c File Reference	106
7.21.1 Macro Definition Documentation	106
7.21.1.1 FALSE	106
7.21.1.2 TRUE	107
7.21.2 Function Documentation	107
7.21.2.1 threadpool_worker()	107
7.22 threadpool.c	107

Index

Chapter 1

Todo List

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	 		 		 					 	 														9
ThreadPool	 		 						 							 								2	3

4 Module Index

Chapter 3

Class Index

3.1 Class List

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

chunk_params	29
ClientInterface	31
Node	35
equest	36
Main structure for managing io_uring and socket events	38
Saurion	11
Structure containing callback functions to handle socket events	16
saurion_wrapper	
ask	50
hreadpool	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/client_interface.hpp	55
/w/saurion/saurion/include/linked_list.h	56
/w/saurion/saurion/include/low_saurion.h	58
/w/saurion/saurion/include/low_saurion_secret.h	66
/w/saurion/saurion/include/saurion.hpp	67
/w/saurion/saurion/include/threadpool.h	67
/w/saurion/src/linked_list.c	68
/w/saurion/src/low_saurion.c	73
/w/saurion/src/main.c	04
/w/saurion/src/saurion.cpp	04
/w/saurion/src/threadpool.c	06

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_callbacks

Structure containing callback functions to handle socket events.

· 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 saurion_set_socket (int p)

Creates a socket.

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		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:

```
struct saurion *s = saurion_create(4);
if (saurion_start(s) != 0) {
    handle_error();
}
saurion_send(s, socket_fd, "Hello, World!");
saurion_stop(s);
saurion_destroy(s);
```

- 1. Create the saurion structure with 4 threads
- 2. Start event processing
- 3. Send a message through a socket
- 4. Stop event processing
- 5. Destroy the structure and free resources

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 108 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 140 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 164 of file low_saurion.c.

```
00167
        if (!iov || !chd_ptr)
00168
00169
           return ERROR_CODE;
00170
00171
        iov->iov_base = malloc (CHUNK_SZ);
00172
       if (!iov->iov_base)
        {
00173
00174
            return ERROR_CODE;
00175
       iov->iov_len = (pos == (amount - 1) ? (size % CHUNK_SZ) : CHUNK_SZ);
if (iov->iov_len == 0)
00176
00177
00178
        {
00179
            iov->iov_len = CHUNK_SZ;
00180
00181 chd_ptr[pos] = iov->iov_base;
00182
        return SUCCESS_CODE;
00183 }
```

5.1.3.2 free_request()

Definition at line 94 of file low saurion.c.

```
00096
        if (children_ptr)
00097
         {
00098
             free (children_ptr);
00099
            children_ptr = NULL;
00100
00101
        for (size_t i = 0; i < amount; ++i)</pre>
00102
00103
             free (req->iov[i].iov_base);
00104
            req->iov[i].iov_base = NULL;
00105
       free (req);
00106
00107
        req = NULL;
       free (children_ptr);
children_ptr = NULL;
00108
00109
00110 }
```

5.1.3.3 initialize_iovec()

```
int initialize_iovec (
    struct iovec * iov,
    size_t amount,
    size_t pos,
    const void * msg,
    size_t size,
    uint8_t h ) [private]
```

Initializes a specified iovec structure with a message fragment.

This function populates the iov_base of the iov_c 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 iov_c , 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 115 of file low_saurion.c.

```
00123
           {
00124
            size_t len = iov->iov_len;
00125
            char *dest = (char *)iov->iov_base;
            char *orig = (char *) msg + pos * CHUNK_SZ;
00126
00127
            size_t cpy_sz = 0;
            if (h)
00128
00129
00130
                 if (pos == 0)
00131
                     uint64_t send_size = htonll (size);
00132
                     memcpy (dest, &send_size, sizeof (uint64_t));
dest += sizeof (uint64_t);
00133
00134
                     len -= sizeof (uint64_t);
00135
00136
00137
                 else
00138
                     orig -= sizeof (uint64_t);
00139
00140
00141
                 if ((pos + 1) == amount)
00142
                   {
00143
                     --len;
                     cpy_sz = (len < size ? len : size);</pre>
00144
00145
                     dest[cpy_sz] = 0;
00146
00147
00148
            cpy_sz = (len < size ? len : size);
00149
            memcpy (dest, orig, cpy_sz);
            dest += cpy_sz;
size_t rem = CHUNK_SZ - (dest - (char *)iov->iov_base);
00150
00151
00152
            memset (dest, 0, rem);
00153
00154
        else
00155
        {
00156
            memset ((char *)iov->iov_base, 0, CHUNK_SZ);
00157
        return SUCCESS CODE:
00158
00159 }
```

5.1.3.4 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 688 of file low_saurion.c.

```
00689 {
         struct chunk_params p;
00690
        p.req = req;
p.dest = dest;
p.len = len;
00691
00692
00693
00694
         if (p.req->iovec_count == 0)
00695
         {
00696
             return ERROR_CODE;
00697
           }
00698
00699
        p.max_iov_cont = calculate_max_iov_content (p.req);
00700
        p.cont_sz = 0;
00701
         p.cont_rem = 0;
         p.curr_iov = 0;
00702
        p.curr_iov_off = 0;
p.dest_off = 0;
p.dest_ptr = NULL;
00703
00704
00706
         if (!prepare_destination (&p))
00707
00708
             return ERROR_CODE;
00709
          }
00710
        uint8_t ok = 1UL;
copy_data (&p, &ok);
00711
00712
00713
00714
         if (validate_and_update (&p, ok))
00715
00716
             return SUCCESS CODE;
00718
        read_chunk_free (&p);
00719
        return ERROR_CODE;
00720 }
```

5.1.3.5 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 835 of file low_saurion.c.

```
00836 {
        LOG_INIT (" ");
00837
00838
        struct saurion *p = (struct saurion *)malloc (sizeof (struct saurion));
00839
        if (!p)
00840
         {
00841
            LOG_END (" ");
00842
            return NULL;
00843
        int ret = 0:
00844
00845
        ret = pthread_mutex_init (&p->status_m, NULL);
00846
        if (ret)
00847
         {
            free (p);
LOG_END ("_");
00848
00849
            return NULL;
00850
00851
00852
        ret = pthread_cond_init (&p->status_c, NULL);
00853
00854
         {
            free (p);
LOG_END (" ");
00855
00856
00857
            return NULL;
00858
00859
        p->m_rings
             = (pthread_mutex_t *) malloc (n_threads * sizeof (pthread_mutex_t));
00860
00861
        if (!p->m_rings)
00862
         {
            free (p);
LOG_END (" ");
00863
00864
00865
            return NULL;
00866
00867
        for (uint32_t i = 0; i < n_threads; ++i)</pre>
00868
            pthread_mutex_init (&(p->m_rings[i]), NULL);
00869
00870
        p->ss=0;
        n_threads = (n_threads < 2 ? 2 : n_threads);
n_threads = (n_threads > NUM_CORES ? NUM_CORES : n_threads);
00872
00873
        p->n_threads = n_threads;
00874
00875
        p->status = 0;
00876
        p->list = NULL;
00877
        p->cb.on_connected = NULL;
00878
        p->cb.on_connected_arg = NULL;
00879
        p->cb.on_readed = NULL;
00880
        p->cb.on_readed_arg = NULL;
00881
        p->cb.on_wrote = NULL;
        p->cb.on_wrote_arg = NULL;
00882
        p->cb.on_closed = NULL;
00883
00884
        p->cb.on_closed_arg = NULL;
00885
        p->cb.on_error = NULL;
00886
        p->cb.on_error_arg = NULL;
        p->next = 0;
p->efds = (int *)malloc (sizeof (int) * p->n_threads);
00887
00888
00889
        if (!p->efds)
00890
```

```
free (p->m_rings);
            free (p);
LOG_END (" ");
00892
00893
00894
            return NULL;
00895
00896
        for (uint32_t i = 0; i < p->n_threads; ++i)
00898
            p->efds[i] = eventfd (0, EFD_NONBLOCK);
00899
            if (p->efds[i] == ERROR_CODE)
00900
              {
00901
                 for (uint32_t j = 0; j < i; ++j)
00902
00903
                     close (p->efds[j]);
00904
00905
                free (p->efds);
00906
                 free (p->m_rings);
                free (p);
LOG_END (" ");
00907
00908
                 return NULL;
00909
00910
00911
00912
        p->rings
00913
            = (struct io_uring *)malloc (sizeof (struct io_uring) * p->n_threads);
        if (!p->rings)
00914
00915
          {
00916
            for (uint32_t j = 0; j < p->n_threads; ++j)
00917
00918
                close (p->efds[j]);
00919
00920
            free (p->efds);
            free (p->m_rings);
00921
            free (p);
LOG_END (" ");
00922
00923
00924
             return NULL;
00925
        for (uint32_t i = 0; i < p->n_threads; ++i)
00926
00927
            memset (&p->rings[i], 0, sizeof (struct io_uring));
00929
            ret = io_uring_queue_init (SAURION_RING_SIZE, &p->rings[i], 0);
00930
00931
              {
00932
                 for (uint32_t j = 0; j < p->n_threads; ++j)
00933
00934
                     close (p->efds[j]);
00935
00936
                free (p->efds);
00937
                free (p->rings);
00938
                 free (p->m_rings);
                free (p);
LOG_END (" ");
00939
00940
00941
                 return NULL;
00942
00943
        p->pool = threadpool_create (p->n_threads);
LOG_END (" ");
00944
00945
00946
        return p;
```

5.1.3.6 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 1189 of file low_saurion.c.

```
01191
        pthread_mutex_lock (&s->status_m);
01192
        while (s->status > 0)
01193
01194
            pthread_cond_wait (&s->status_c, &s->status_m);
01195
01196
        pthread_mutex_unlock (&s->status_m);
01197
        threadpool_destroy (s->pool);
01198
        for (uint32_t i = 0; i < s->n_threads; ++i)
01199
01200
            io_uring_queue_exit (&s->rings[i]);
01201
            pthread_mutex_destroy (&s->m_rings[i]);
01202
01203
        free (s->m_rings);
01204
        list_free (&s->list);
        for (uint32_t i = 0; i < s->n_threads; ++i)
01205
01206
01207
            close (s->efds[i]);
01208
01209
        free (s->efds);
01210
        if (!s->ss)
        {
01211
01212
           close (s->ss);
01213
01214
       free (s->rings);
01215
        pthread_mutex_destroy (&s->status_m);
01216
       pthread_cond_destroy (&s->status_c);
01217
        free (s);
01218 }
```

5.1.3.7 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 sent.	
msg	Pointer to the character string (message) to be sent.	

```
Definition at line 1222 of file low_saurion.c.
```

```
01223 {
01224 add_write (s, fd, msg, next (s));
01225 }
```

5.1.3.8 saurion set socket()

```
\label{eq:control_set_socket} \mbox{ int saurion\_set\_socket (} \\ \mbox{ int } p \mbox{ )}
```

Creates a socket.

Creates and sets a socket, ready for saurion configuration.

Parameters

```
p port
```

Returns

result of socket creation.

Definition at line 797 of file low_saurion.c.

```
00798 {
        int sock = 0;
00800
        struct sockaddr_in srv_addr;
00801
00802
        sock = socket (PF_INET, SOCK_STREAM, 0);
00803
        if (sock < 1)
00804
00805
            return ERROR_CODE;
00806
         }
00807
80800
        int enable = 1;
00809
        if (setsockopt (sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof (int)) < 0)</pre>
00810
00811
            return ERROR_CODE;
00812
          }
00813
00814
        memset (&srv_addr, 0, sizeof (srv_addr));
        srv_addr.sin_family = AF_INET;
srv_addr.sin_port = htons (p);
00815
00816
00817
        srv_addr.sin_addr.s_addr = htonl (INADDR_ANY);
00818
00819
        if (bind (sock, (const struct sockaddr *)&srv_addr, sizeof (srv_addr)) < 0)</pre>
00821
            return ERROR_CODE;
00822
00823
        if (listen (sock, ACCEPT_QUEUE) < 0)</pre>
00824
00825
00826
            return ERROR_CODE;
00827
00828
00829
       return sock;
00830 }
```

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 1147 of file low_saurion.c.

```
01148 {
01149
        threadpool_init (s->pool);
01150
        threadpool_add (s->pool, saurion_worker_master, s);
01151
        struct saurion_wrapper *ss = NULL;
01152
        for (uint32_t i = 1; i < s->n_threads; ++i)
01153
01154
           ss = (struct saurion_wrapper *)malloc (sizeof (struct saurion_wrapper));
01155
            if (!ss)
            {
            return ERROR_CODE;
}
01156
01157
01158
           ss->s = s;
ss->sel = i;
01159
01160
01161
           threadpool_add (s->pool, saurion_worker_slave, ss);
01162
01163 pthread_mutex_lock (&s->status_m);
01164
        while (s->status < (int)s->n_threads)
01165
01166
           pthread_cond_wait (&s->status_c, &s->status_m);
01168 pthread_mutex_unlock (&s->status_m);
01169
        return SUCCESS_CODE;
01170 }
```

5.1.3.10 saurion_stop()

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 1174 of file low saurion.c.

5.1.3.11 set_request()

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 188 of file low saurion.c.

```
00190
00191
        uint64_t full_size = s;
00192
        if (h)
00193
          {
00194
             full_size += (sizeof (uint64_t) + sizeof (uint8_t));
00195
        size_t amount = full_size / CHUNK_SZ;
00196
        amount = amount + (full_size % CHUNK_SZ == 0 ? 0 : 1);
00197
        struct request *temp = (struct request *) malloc (
00198
00199
             sizeof (struct request) + sizeof (struct iovec) * amount);
00200
         if (!temp)
00201
00202
             return ERROR_CODE;
           }
00203
00204
        if (!*r)
00205
          {
             *r = temp;
00206
             (*r)->prev = NULL;
00207
00208
             (*r)->prev_size = 0;
00209
             (*r) \rightarrow prev\_remain = 0;
00210
             (*r) \rightarrow next_iov = 0;
00211
             (*r)->next_offset = 0;
00212
00213
        else
00214
00215
             temp->client_socket = (*r)->client_socket;
             temp >errent_steket = (*r) >errent_s
temp->event_type = (*r) ->event_type;
temp->prev = (*r) ->prev;
00216
00217
00218
             temp->prev_size = (*r)->prev_size;
00219
             temp->prev_remain = (*r)->prev_remain;
00220
             temp->next_iov = (*r)->next_iov;
00221
             temp->next_offset = (*r)->next_offset;
00222
             *r = temp;
00223
00224
        struct request *req = *r;
```

5.2 ThreadPool 23

```
req->iovec_count = (int)amount;
00226
        void **children_ptr = (void **)malloc (amount * sizeof (void *));
00227
        if (!children_ptr)
00228
            free_request (req, children_ptr, 0);
00229
00230
            return ERROR_CODE;
00231
00232
        for (size_t i = 0; i < amount; ++i)</pre>
00233
00234
            if (!allocate_iovec (&req->iov[i], amount, i, full_size, children_ptr))
00235
00236
               free_request (req, children_ptr, amount);
00237
                return ERROR_CODE;
00238
00239
            if (!initialize_iovec (&req->iov[i], amount, i, m, s, h))
00240
                free_request (req, children_ptr, amount);
00241
00242
                return ERROR_CODE;
00244
00245
       if (!list_insert (l, req, amount, children_ptr))
00246
00247
            free_request (req, children_ptr, amount);
00248
            return ERROR_CODE;
00249
00250 free (children_ptr);
00251
       return SUCCESS_CODE;
00252 }
```

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()

LOG_END (" ");

return;

if (pool == NULL || function == NULL)

00172 00173 00174

00175

```
00176
00177
00178
         struct task *new_task = malloc (sizeof (struct task));
00179
         if (new_task == NULL)
00180
00181
              LOG_END (" ");
00182
              return;
00183
00184
         new_task->function = function;
new_task->argument = argument;
00185
00186
         new_task->next = NULL;
00187
00188
00189
         pthread_mutex_lock (&pool->queue_lock);
00190
00191
         if (pool->task_queue_head == NULL)
00192
              pool->task_queue_head = new_task;
pool->task_queue_tail = new_task;
00193
00194
00195
00196
00197
              pool->task_queue_tail->next = new_task;
pool->task_queue_tail = new_task;
00198
00199
00200
00201
         pthread_cond_signal (&pool->queue_cond);
00202
00203
         pthread_mutex_unlock (&pool->queue_lock);
00204
         LOG_END (" ");
00205 }
```

5.2.2.2 threadpool create()

Definition at line 31 of file threadpool.c.

```
00032 {
00033
        LOG_INIT (" ");
        struct threadpool *pool = malloc (sizeof (struct threadpool));
00034
00035
        if (pool == NULL)
00036
            LOG_END (" ");
00037
00038
            return NULL;
00039
00040
        if (num_threads < 3)</pre>
00041
        {
00042
            num_threads = 3;
00043
00044
        if (num_threads > NUM_CORES)
00045
00046
           num_threads = NUM_CORES;
00047
00048
00049
        pool->num_threads = num_threads;
00050
        pool->threads = malloc (sizeof (pthread_t) * num_threads);
00051
        if (pool->threads == NULL)
00052
         {
00053
            free (pool);
            LOG_END (" ");
00054
00055
            return NULL;
00056
00057
00058
        pool->task_queue_head = NULL;
        pool->task_queue_tail = NULL;
00059
00060
        pool->stop = FALSE;
00061
        pool->started = FALSE;
00062
        if (pthread_mutex_init (&pool->queue_lock, NULL) != 0)
00063
00064
00065
            free (pool->threads);
00066
            free (pool);
00067
            LOG_END (" ");
00068
            return NULL;
00069
00070
00071
        if (pthread_cond_init (&pool->queue_cond, NULL) != 0)
        {
```

5.2 ThreadPool 25

```
pthread_mutex_destroy (&pool->queue_lock);
00074
               free (pool->threads);
              free (pool);
LOG_END (" ");
00075
00076
00077
               return NULL;
00078
00079
08000
          if (pthread_cond_init (&pool->empty_cond, NULL) != 0)
00081
              pthread_mutex_destroy (&pool->queue_lock);
pthread_cond_destroy (&pool->queue_cond);
free (pool->threads);
00082
00083
00084
              free (pool);
LOG_END (" ");
00085
00086
00087
               return NULL;
00088
00089
00090
         LOG_END (" ");
         return pool;
00092 }
```

5.2.2.3 threadpool_create_default()

Definition at line 95 of file threadpool.c.

```
00096 {
00097    return threadpool_create (NUM_CORES);
00098 }
```

5.2.2.4 threadpool destroy()

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

Definition at line 266 of file threadpool.c.

```
00267 {
00268
         LOG_INIT (" ");
         if (pool == NULL)
00269
00270
00271
               LOG_END (" ");
00272
              return;
00273
00274 threadpool_stop (pool);
00275
00276
        pthread_mutex_destroy (&pool->queue_lock);
        pthread_cond_destroy (&pool->queue_cond);
pthread_cond_destroy (&pool->empty_cond);
00277
00278
00279
         free (pool->threads);
free (pool);
LOG_END (" ");
00280
00281
00282
00283 }
```

5.2.2.5 threadpool_empty()

Definition at line 232 of file threadpool.c.

```
00233 {
         LOG INIT (" ");
00234
00235
         if (pool == NULL)
00236
00237
             LOG_END (" ");
             return TRUE;
00238
00239
        pthread_mutex_lock (&pool->queue_lock);
int empty = (pool->task_queue_head == NULL);
00240
00241
00242
        pthread_mutex_unlock (&pool->queue_lock);
00243
         LOG_END (" ");
00244
         return empty;
00245 }
```

5.2.2.6 threadpool_init()

Definition at line 145 of file threadpool.c.

```
00146 {
00147
        LOG_INIT (" ");
00148
        if (pool == NULL || pool->started)
00149
            LOG_END (" ");
00150
00151
            return;
00152
        for (size_t i = 0; i < pool->num_threads; i++)
00153
00154
00155
            if (pthread_create (&pool->threads[i], NULL, threadpool_worker,
00156
                                  (void *)pool)
00157
                ! = 0)
00158
              {
                pool->stop = TRUE;
00159
00160
                break;
00161
              }
00162
       pool->started = TRUE;
LOG_END (" ");
00163
00164
00165 }
```

5.2.2.7 threadpool_stop()

Definition at line 208 of file threadpool.c.

```
00209 {
00210
       LOG_INIT (" ");
       if (pool == NULL || !pool->started)
00211
00212
        {
00213
           LOG_END (" ");
00214
           return;
00215
00216
       threadpool_wait_empty (pool);
00217
00218
       pthread_mutex_lock (&pool->queue_lock);
00219
       pool->stop = TRUE;
       pthread_cond_broadcast (&pool->queue_cond);
       pthread_mutex_unlock (&pool->queue_lock);
```

5.2 ThreadPool 27

5.2.2.8 threadpool_wait_empty()

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

Definition at line 248 of file threadpool.c.

```
00249 {
00250
          LOG_INIT (" ");
          if (pool == NULL)
00251
00252
              LOG_END (" ");
00253
          __G_END
    return;
}
00254
00255
00256
         pthread_mutex_lock (&pool->queue_lock);
while (pool->task_queue_head != NULL)
00258
00259
              pthread_cond_wait (&pool->empty_cond, &pool->queue_lock);
00260
00261
         pthread_mutex_unlock (&pool->queue_lock);
LOG_END (" ");
00262
00263 }
```

28 Module Documentation

Chapter 6

Class Documentation

6.1 chunk_params Struct Reference

Collaboration diagram for chunk_params:

Public Attributes

- void ** dest
- void * dest_ptr
- size_t dest_off
- struct request * req
- size_t cont_sz
- size_t cont_rem
- size_t max_iov_cont
- size_t curr_iov
- size_t curr_iov_off
- size_t * len

6.1.1 Detailed Description

Definition at line 452 of file low_saurion.c.

6.1.2 Member Data Documentation

6.1.2.1 cont_rem

```
size_t chunk_params::cont_rem
```

Definition at line 459 of file low_saurion.c.

6.1.2.2 cont_sz

size_t chunk_params::cont_sz

Definition at line 458 of file low_saurion.c.

6.1.2.3 curr_iov

size_t chunk_params::curr_iov

Definition at line 461 of file low_saurion.c.

6.1.2.4 curr_iov_off

size_t chunk_params::curr_iov_off

Definition at line 462 of file low_saurion.c.

6.1.2.5 dest

void** chunk_params::dest

Definition at line 454 of file low_saurion.c.

6.1.2.6 dest off

size_t chunk_params::dest_off

Definition at line 456 of file low_saurion.c.

6.1.2.7 dest_ptr

void* chunk_params::dest_ptr

Definition at line 455 of file low_saurion.c.

6.1.2.8 len

```
size_t* chunk_params::len
```

Definition at line 463 of file low_saurion.c.

6.1.2.9 max_iov_cont

```
size_t chunk_params::max_iov_cont
```

Definition at line 460 of file low_saurion.c.

6.1.2.10 req

```
struct request* chunk_params::req
```

Definition at line 457 of file low_saurion.c.

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

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

6.2 ClientInterface Class Reference

```
#include <client_interface.hpp>
```

Public Member Functions

- ClientInterface () noexcept
- ∼ClientInterface ()
- ClientInterface (const ClientInterface &)=delete
- ClientInterface (ClientInterface &&)=delete
- ClientInterface & operator= (const ClientInterface &)=delete
- ClientInterface & operator= (ClientInterface &&)=delete
- void connect (const uint n)
- void disconnect ()
- void send (const uint n, const char *const msg, uint delay)
- uint64_t reads (const std::string &search) const
- · void clean () const
- std::string getFifoPath () const
- int getPort () const

Private Attributes

```
pid_t pid
FILE * fifo
std::string fifoname = set_fifoname ()
int port = set_port ()
```

6.2.1 Detailed Description

Definition at line 17 of file client_interface.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 ClientInterface() [1/3]

```
ClientInterface::ClientInterface ( ) [explicit], [noexcept]
```

6.2.2.2 ∼ClientInterface()

```
ClientInterface::~ClientInterface ( )
```

6.2.2.3 ClientInterface() [2/3]

6.2.2.4 ClientInterface() [3/3]

6.2.3 Member Function Documentation

6.2.3.1 clean()

```
void ClientInterface::clean ( ) const
```

6.2.3.2 connect()

6.2.3.3 disconnect()

```
void ClientInterface::disconnect ( )
```

6.2.3.4 getFifoPath()

```
std::string ClientInterface::getFifoPath ( ) const
```

6.2.3.5 getPort()

```
int ClientInterface::getPort ( ) const
```

6.2.3.6 operator=() [1/2]

6.2.3.7 operator=() [2/2]

6.2.3.8 reads()

6.2.3.9 send()

6.2.4 Member Data Documentation

6.2.4.1 fifo

```
FILE* ClientInterface::fifo [private]
```

Definition at line 40 of file client_interface.hpp.

6.2.4.2 fifoname

```
std::string ClientInterface::fifoname = set_fifoname () [private]
```

Definition at line 41 of file client_interface.hpp.

6.2.4.3 pid

```
pid_t ClientInterface::pid [private]
```

Definition at line 39 of file client_interface.hpp.

6.2.4.4 port

```
int ClientInterface::port = set_port () [private]
```

Definition at line 42 of file client_interface.hpp.

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

/__w/saurion/saurion/include/client_interface.hpp

6.3 Node Struct Reference 35

6.3 Node Struct Reference

Collaboration diagram for Node:

Public Attributes

- void * ptr
- size t size
- struct Node ** children
- struct Node * next

6.3.1 Detailed Description

Definition at line 7 of file linked_list.c.

6.3.2 Member Data Documentation

6.3.2.1 children

struct Node** Node::children

Definition at line 11 of file linked_list.c.

6.3.2.2 next

struct Node* Node::next

Definition at line 12 of file linked_list.c.

6.3.2.3 ptr

void* Node::ptr

Definition at line 9 of file linked_list.c.

6.3.2.4 size

```
size_t Node::size
```

Definition at line 10 of file linked_list.c.

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

• /__w/saurion/saurion/src/linked_list.c

6.4 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 []

6.4.1 Detailed Description

Definition at line 32 of file low_saurion.c.

6.4.2 Member Data Documentation

6.4.2.1 client_socket

```
int request::client_socket
```

Definition at line 41 of file low_saurion.c.

6.4.2.2 event_type

```
int request::event_type
```

Definition at line 39 of file low_saurion.c.

6.4.2.3 iov

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

Definition at line 42 of file low_saurion.c.

6.4.2.4 iovec_count

```
size_t request::iovec_count
```

Definition at line 40 of file low_saurion.c.

6.4.2.5 next_iov

```
size_t request::next_iov
```

Definition at line 37 of file low_saurion.c.

6.4.2.6 next_offset

```
size_t request::next_offset
```

Definition at line 38 of file low_saurion.c.

6.4.2.7 prev

```
void* request::prev
```

Definition at line 34 of file low_saurion.c.

6.4.2.8 prev_remain

```
size_t request::prev_remain
```

Definition at line 36 of file low_saurion.c.

6.4.2.9 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

6.5 saurion Struct Reference

Main structure for managing io_uring and socket events.

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

Collaboration diagram for saurion:

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
- · struct saurion callbacks cb

6.5.1 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 214 of file low_saurion.h.

6.5.2 Member Data Documentation

6.5.2.1 cb

struct saurion_callbacks saurion::cb

Definition at line 239 of file low_saurion.h.

6.5.2.2 efds

int* saurion::efds

Eventfd descriptors used for internal signaling between threads.

Definition at line 223 of file low_saurion.h.

6.5.2.3 list

struct Node* saurion::list

Linked list for storing active requests.

Definition at line 225 of file low_saurion.h.

6.5.2.4 m_rings

pthread_mutex_t* saurion::m_rings

Array of mutexes to protect the io_uring rings.

Definition at line 219 of file low_saurion.h.

6.5.2.5 n_threads

uint32_t saurion::n_threads

Number of threads in the thread pool.

Definition at line 235 of file low_saurion.h.

6.5.2.6 next

```
uint32_t saurion::next
```

Index of the next io_uring ring to which an event will be added.

Definition at line 237 of file low saurion.h.

6.5.2.7 pool

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

Thread pool for executing tasks in parallel.

Definition at line 233 of file low_saurion.h.

6.5.2.8 rings

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

Array of io_uring structures for managing the event queue.

Definition at line 217 of file low_saurion.h.

6.5.2.9 ss

int saurion::ss

Server socket descriptor for accepting connections.

Definition at line 221 of file low_saurion.h.

6.5.2.10 status

int saurion::status

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

Definition at line 231 of file low_saurion.h.

6.5.2.11 status_c

```
pthread_cond_t saurion::status_c
```

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

Definition at line 229 of file low_saurion.h.

6.5.2.12 status_m

```
pthread_mutex_t saurion::status_m
```

Mutex to protect the state of the structure.

Definition at line 227 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 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

6.6.1 Detailed Description

Definition at line 7 of file saurion.hpp.

6.6.2 Member Typedef Documentation

6.6.2.1 ClosedCb

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

Definition at line 14 of file saurion.hpp.

6.6.2.2 ConnectedCb

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

Definition at line 10 of file saurion.hpp.

6.6.2.3 ErrorCb

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

Definition at line 15 of file saurion.hpp.

6.6.2.4 ReadedCb

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

Definition at line 11 of file saurion.hpp.

6.6.2.5 WroteCb

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

Definition at line 13 of file saurion.hpp.

6.6.3 Constructor & Destructor Documentation

6.6.3.1 Saurion() [1/3]

Definition at line 6 of file saurion.cpp.

```
00007 {
00008 this->s = saurion_create (thds);
00009 if (!this->s)
00010 {
00011 return;
00012 }
00013 this->s->ss = sck;
00014 }
```

6.6.3.2 ∼Saurion()

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

Definition at line 16 of file saurion.cpp.

```
00017 {
00018 close (s->ss);
00019 saurion_destroy (this->s);
00020 }
```

6.6.3.3 Saurion() [2/3]

6.6.3.4 Saurion() [3/3]

6.6.4 Member Function Documentation

6.6.4.1 init()

6.6.4.2 on_closed()

Definition at line 62 of file saurion.cpp.

6.6.4.3 on_connected()

Definition at line 38 of file saurion.cpp.

6.6.4.4 on_error()

Definition at line 70 of file saurion.cpp.

```
00071 {
00072    s->cb.on_error = ncb;
00073    s->cb.on_error_arg = arg;
00074    return this;
00075 }
```

6.6.4.5 on_readed()

Definition at line 46 of file saurion.cpp.

6.6.4.6 on_wrote()

Definition at line 54 of file saurion.cpp.

6.6.4.7 operator=() [1/2]

6.6.4.8 operator=() [2/2]

6.6.4.9 send()

Definition at line 78 of file saurion.cpp.

6.6.4.10 stop()

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

Definition at line 32 of file saurion.cpp.
00033 {
00034    saurion_stop (this->s);
00035 }
```

6.6.5 Member Data Documentation

6.6.5.1 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

6.7 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

6.7.1 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 149 of file low_saurion.h.

6.7.2 Member Data Documentation

6.7.2.1 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 189 of file low_saurion.h.

6.7.2.2 on_closed_arg

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

Additional argument for the close callback.

Definition at line 191 of file low_saurion.h.

6.7.2.3 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.

Generated by Doxygen

Definition at line 157 of file low_saurion.h.

6.7.2.4 on_connected_arg

void* saurion_callbacks::on_connected_arg

Additional argument for the connection callback.

Definition at line 159 of file low_saurion.h.

6.7.2.5 on_error

 $\label{local_const_con$

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 201 of file low saurion.h.

6.7.2.6 on_error_arg

void* saurion_callbacks::on_error_arg

Additional argument for the error callback.

Definition at line 204 of file low_saurion.h.

6.7.2.7 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 169 of file low_saurion.h.

6.7.2.8 on_readed_arg

void* saurion_callbacks::on_readed_arg

Additional argument for the read callback.

Definition at line 172 of file low_saurion.h.

6.7.2.9 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 180 of file low_saurion.h.

6.7.2.10 on_wrote_arg

void* saurion_callbacks::on_wrote_arg

Additional argument for the write callback.

Definition at line 181 of file low_saurion.h.

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

• /__w/saurion/saurion/include/low_saurion.h

6.8 saurion_wrapper Struct Reference

Collaboration diagram for saurion_wrapper:

Public Attributes

- struct saurion * s
- uint32_t sel

6.8.1 Detailed Description

Definition at line 50 of file low_saurion.c.

6.8.2 Member Data Documentation

6.8.2.1 s

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

Definition at line 52 of file low_saurion.c.

6.8.2.2 sel

```
uint32_t saurion_wrapper::sel
```

Definition at line 53 of file low_saurion.c.

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

• /__w/saurion/saurion/src/low_saurion.c

6.9 task Struct Reference

Collaboration diagram for task:

Public Attributes

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

6.9.1 Detailed Description

Definition at line 10 of file threadpool.c.

6.9.2 Member Data Documentation

6.9.2.1 argument

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

Definition at line 13 of file threadpool.c.

6.9.2.2 function

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

Definition at line 12 of file threadpool.c.

6.9.2.3 next

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

Definition at line 14 of file threadpool.c.

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

• /__w/saurion/saurion/src/threadpool.c

6.10 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

6.10.1 Detailed Description

Definition at line 17 of file threadpool.c.

6.10.2 Member Data Documentation

6.10.2.1 empty_cond

```
pthread_cond_t threadpool::empty_cond
```

Definition at line 25 of file threadpool.c.

6.10.2.2 num threads

```
size_t threadpool::num_threads
```

Definition at line 20 of file threadpool.c.

6.10.2.3 queue_cond

```
pthread_cond_t threadpool::queue_cond
```

Definition at line 24 of file threadpool.c.

6.10.2.4 queue_lock

```
pthread_mutex_t threadpool::queue_lock
```

Definition at line 23 of file threadpool.c.

6.10.2.5 started

int threadpool::started

Definition at line 27 of file threadpool.c.

6.10.2.6 stop

int threadpool::stop

Definition at line 26 of file threadpool.c.

6.10.2.7 task_queue_head

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

Definition at line 21 of file threadpool.c.

6.10.2.8 task_queue_tail

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

Definition at line 22 of file threadpool.c.

6.10.2.9 threads

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

Definition at line 19 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/client_interface.hpp File Reference

```
#include <cstdint>
#include <cstdio>
#include <string>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
Include dependency graph for client_interface.hpp:
```

include dependency graph for cheft_interface.ripp.

Classes

• class ClientInterface

Functions

- int set_port ()
- std::string set_fifoname ()

7.1.1 Function Documentation

7.1.1.1 set_fifoname()

```
std::string set_fifoname ( )
```

7.1.1.2 set_port()

```
int set_port ( )
```

56 File Documentation

7.2 client interface.hpp

Go to the documentation of this file. 00001 #ifndef CLIENT_INTERFACE_HPP 00002 #define CLIENT_INTERFACE_HPP 00003 00004 #include <cstdint> 00005 #include <cstdio> 00006 #include <string> 00007 #include <sys/stat.h> 00008 #include <sys/types.h> 00009 #include <unistd.h> 00010 00011 // set_port 00012 int set_port (); 00013 00014 // set fifoname 00015 std::string set_fifoname (); 00016 00017 class ClientInterface 00018 { 00019 public: 00020 explicit ClientInterface () noexcept; 00021 ~ClientInterface (); 00022 ClientInterface (const ClientInterface &) = delete; 00024 ClientInterface (ClientInterface &&) = delete; 00025 ClientInterface & operator= (const ClientInterface &) = delete; 00026 ClientInterface &operator= (ClientInterface &&) = delete; 00027 00028 void connect (const uint n); 00029 void disconnect (); 00030 00031 void send (const uint n, const char *const msg, uint delay); 00032 uint64_t reads (const std::string &search) const; 00033 void clean () const; 00034 00035 std::string getFifoPath () const; 00036 int getPort () const; 00037 00038 private: 00039 pid_t pid; 00040 FILE *fifo; std::string fifoname = set_fifoname (); 00042 int port = set_port ();

7.3 /_w/saurion/saurion/include/linked_list.h File Reference

#include <stddef.h>

00045 #endif // !CLIENT_INTERFACE_HPP

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

Functions

00043 }; 00044

- 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)

7.3.1 Function Documentation

7.3.1.1 list_delete_node()

```
void list_delete_node (
              struct Node ** head,
              const void *const ptr )
Definition at line 109 of file linked_list.c.
00110 {
00111
        pthread_mutex_lock (&list_mutex);
        struct Node *current = *head;
00112
00113
       struct Node *prev = NULL;
00114
00115
       if (current && current->ptr == ptr)
00116
           *head = current->next;
00117
00118
           free node (current);
00119
           pthread_mutex_unlock (&list_mutex);
00120
           return;
00121
00122
       while (current && current->ptr != ptr)
00123
00124
        {
00125
           prev = current;
00126
           current = current->next;
        }
00127
00128
00129
       if (!current)
       {
00130
00131
           pthread_mutex_unlock (&list_mutex);
00132
           return;
00133
00134
00135 prev->next = current->next;
00136
       free_node (current);
00137
       pthread_mutex_unlock (&list_mutex);
```

7.3.1.2 list_free()

00138 }

Definition at line 141 of file linked_list.c.

```
00142 {
00143
         pthread_mutex_lock (&list_mutex);
         struct Node *current = *head;
struct Node *next;
00144
00145
00146
00147
         while (current)
00148
00149
             next = current->next;
              free_node (current);
current = next;
00150
00151
00152
00153
00154
         *head = NULL;
00155    pthread_mutex_unlock (&list_mutex);
00156 }
```

7.3.1.3 list insert()

58 File Documentation

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

Definition at line 68 of file linked list.c.

```
00069
00070
        struct Node *new_node = create_node (ptr, amount, children);
        if (!new_node)
00072
00073
            return ERROR_CODE;
00074
00075
        pthread_mutex_lock (&list_mutex);
00076
       if (!*head)
{
00077
00078
            *head = new_node;
00079
           pthread_mutex_unlock (&list_mutex);
00080
            return SUCCESS_CODE;
00081
       struct Node *temp = *head;
00082
00083
       while (temp->next)
00084
00085
            temp = temp->next;
00086
       temp->next = new_node;
00087
00088
       pthread_mutex_unlock (&list_mutex);
00089
        return SUCCESS_CODE;
00090 }
```

7.4 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
       [[nodiscard]]
00014
       int list_insert (struct Node **head, void *ptr, size_t amount,
00015
                          void **children);
00016
00017
        void list_delete_node (struct Node **head, const void *const ptr);
00018
00019
       void list free (struct Node **head);
00020
00021 #ifdef __cplusplus
00022 }
00023 #endif
00024
00025 #endif // !LINKED_LIST_H
```

7.5 / 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 callbacks

Structure containing callback functions to handle socket events.

· 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 saurion_set_socket (int p)

Creates a socket.

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.

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
- struct saurion_callbacks cb

60 File Documentation

7.5.1 Variable Documentation

7.5.1.1 cb

```
struct saurion_callbacks cb
```

Definition at line 23 of file low_saurion.h.

7.5.1.2 efds

int* efds

Eventfd descriptors used for internal signaling between threads.

Definition at line 7 of file low_saurion.h.

7.5.1.3 list

```
struct Node* list
```

Linked list for storing active requests.

Definition at line 9 of file low_saurion.h.

7.5.1.4 m_rings

```
pthread_mutex_t* m_rings
```

Array of mutexes to protect the io_uring rings.

Definition at line 3 of file low_saurion.h.

7.5.1.5 n_threads

uint32_t n_threads

Number of threads in the thread pool.

Definition at line 19 of file low_saurion.h.

7.5.1.6 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.

7.5.1.7 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.

7.5.1.8 on_closed_arg

```
void* on_closed_arg
```

Additional argument for the close callback.

Definition at line 40 of file low_saurion.h.

7.5.1.9 on connected

Callback for handling new connections.

Parameters

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

62 File Documentation

Definition at line 6 of file low_saurion.h.

7.5.1.10 on_connected_arg

```
void* on_connected_arg
```

Additional argument for the connection callback.

Definition at line 8 of file low_saurion.h.

7.5.1.11 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.

7.5.1.12 on_error_arg

```
void* on_error_arg
```

Additional argument for the error callback.

Definition at line 53 of file low_saurion.h.

7.5.1.13 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.

7.5.1.14 on_readed_arg

```
void* on_readed_arg
```

Additional argument for the read callback.

Definition at line 21 of file low_saurion.h.

7.5.1.15 on_wrote

Callback for handling write events.

Parameters

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

Definition at line 29 of file low_saurion.h.

7.5.1.16 on_wrote_arg

```
void* on_wrote_arg
```

Additional argument for the write callback.

Definition at line 30 of file low_saurion.h.

7.5.1.17 pool

```
struct threadpool* pool
```

Thread pool for executing tasks in parallel.

Definition at line 17 of file low_saurion.h.

7.5.1.18 rings

```
struct io_uring* rings
```

Array of io_uring structures for managing the event queue.

Definition at line 1 of file low_saurion.h.

7.5.1.19 ss

int ss

Server socket descriptor for accepting connections.

Definition at line 5 of file low_saurion.h.

7.5.1.20 status

int status

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

Definition at line 15 of file low_saurion.h.

7.5.1.21 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.

7.6 low_saurion.h

7.5.1.22 status_m

```
pthread_mutex_t status_m
```

Mutex to protect the state of the structure.

Definition at line 11 of file low saurion.h.

7.6 low_saurion.h

Go to the documentation of this file.

```
00105 #ifndef LOW_SAURION_H
00106 #define LOW_SAURION_H
00108 #define _POSIX_C_SOURCE 200809L
00109
00110 #include <pthread.h>
                              // for pthread_mutex_t, pthread_cond_t
00111 #include <stdint.h> // for wint32_t
00112 #include <sys/types.h> // for ssize_t
00113
00114 #ifdef __cplusplus
00115 extern "C"
00116 {
00117 #endif
00118
00140 #define PACKING_SZ 32
00149
       struct saurion_callbacks
00150
00157
          void (*on_connected) (const int fd, void *arg);
00159
          void *on_connected_arg;
00160
          void (*on_readed) (const int fd, const void *const content,
00169
00170
                              const ssize_t len, void *arg);
00172
          void *on_readed_arg;
00173
          void (*on_wrote) (const int fd, void *arg);
00180
00181
          void *on_wrote_arg;
void (*on_closed) (const int fd, void *arg);
00189
00191
          void *on_closed_arg;
00192
00201
          void (*on_error) (const int fd, const char *const content,
00202
                             const ssize_t len, void *arg);
00204
          void *on_error_arg;
00205
       } __attribute__ ((aligned (PACKING_SZ)));
00206
00214
        struct saurion
00215
00217
         struct io_uring *rings;
00219
          pthread_mutex_t *m_rings;
00221
          int ss;
00223
          int *efds;
          struct Node *list;
00225
00227
          pthread_mutex_t status_m;
00229
          pthread_cond_t status_c;
00231
          int status:
00233
          struct threadpool *pool;
00235
          uint32_t n_threads;
00237
          uint32_t next;
00238
00239
         struct saurion_callbacks cb;
00240
       } __attribute__ ((aligned (PACKING_SZ)));
00241
00251
        int saurion set socket (int p);
00252
00265
        [[nodiscard]]
00266
        struct saurion *saurion_create (uint32_t n_threads);
00267
00280
        [[nodiscard]]
00281
        int saurion_start (struct saurion *s);
00282
        void saurion_stop (const struct saurion *s);
00293
00294
00307
        void saurion_destroy (struct saurion *s);
00308
00321
        void saurion_send (struct saurion *s, const int fd, const char *const msg);
00322
```

```
00323 #ifdef __cplusplus

00324 }

00325 #endif

00326

00327 #endif // !LOW_SAURION_H

00328
```

7.7 / 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 lovec (struct lovec *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)

7.8 low_saurion_secret.h

Go to the documentation of this file.

```
00001 #ifndef LOW_SAURION_SECRET_
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;
       size_t prev_size;
size_t prev_remain;
size_t next_iov;
00019
00020
00021
00022
       size_t next_offset;
00023
        int event_type;
00024
       size_t iovec_count;
00025
       int client socket:
00026
       struct iovec iov[];
00027 };
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 *msg, 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
00176 #endif // !LOW_SAURION_SECRET_H
```

7.9 / 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

7.10 saurion.hpp

Go to the documentation of this file.

```
00001 #ifndef SAURION HPF
00002 #define SAURION_HPP
00004 #include <stdint.h>
                            // for uint32_t
00005 #include <sys/types.h> // for ssize_t
00006
00007 class Saurion
00008 {
00009 public:
00010 using ConnectedCb = void (*) (const int, void *);
00011
       using ReadedCb
00012
           = void (*) (const int, const void *const, const ssize_t, void *);
       using WroteCb = void (*) (const int, void *);
00013
       using ClosedCb = void (*) (const int, void *);
00014
00015
       using ErrorCb
            = void (*) (const int, const char *const, const ssize_t, void *);
00017
00018
       explicit Saurion (const uint32_t thds, const int sck) noexcept;
00019
       ~Saurion ();
00020
00021
       Saurion (const Saurion &) = delete;
       Saurion (Saurion &&) = delete;
00023
       Saurion & operator = (const Saurion &) = delete;
00024 Saurion & operator= (Saurion & &) = delete;
00025
00026
00027
       void init () noexcept;
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;
       Saurion *on_closed (ClosedCb ncb, void *arg) noexcept;
00032
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 // !SAURION_HPP
```

7.11 /__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)

7.12 threadpool.h

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

```
00001
00006 #ifndef THREADPOOL H
00007 #define THREADPOOL_H
80000
00009 #include <stddef.h> // for size_t
00010
00011 #ifdef __cplusplus
00012 extern "C"
00013 {
00014 #endif
00015
00016
       struct threadpool;
00017
00018
       struct threadpool *threadpool_create (size_t num_threads);
00019
00020
       struct threadpool *threadpool_create_default (void);
00021
00022
       void threadpool_init (struct threadpool *pool);
00023
        void threadpool_add (struct threadpool *pool, void (*function) (void *),
00024
00025
                              void *argument);
00026
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
```

7.13 /_w/saurion/saurion/src/linked_list.c File Reference

```
#include "linked_list.h"
#include "config.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

7.13.1 Function Documentation

7.13.1.1 create_node()

Definition at line 19 of file linked_list.c.

```
00020 {
00021
         struct Node *new_node = (struct Node *)malloc (sizeof (struct Node));
00022
        if (!new_node)
00023
         {
00024
            return NULL;
00025
00026
        new_node->ptr = ptr;
00027
        new_node->size = amount;
        new_node->children = NULL;
00028
00029
        if (amount <= 0)
00030
          {
00031
            new_node->next = NULL;
00032
             return new_node;
00033
00034
        new_node->children
00035
             = (struct Node **) malloc (sizeof (struct Node *) * amount);
        if (!new_node->children)
00036
00037
          {
00038
             free (new_node);
00039
             return NULL;
00040
        for (size_t i = 0; i < amount; ++i)</pre>
00041
00042
             new_node->children[i] = (struct Node *)malloc (sizeof (struct Node));
00044
             if (!new_node->children[i])
00045
00046
00047
                 for (size_t j = 0; j < i; ++j)
00048
00049
                     free (new_node->children[j]);
00050
00051
                 free (new_node);
00052
                 return NULL;
00053
00054
00055
        for (size_t i = 0; i < amount; ++i)</pre>
00056
00057
             new_node->children[i]->size = 0;
             new_node->children[i]->next = NULL;
new_node->children[i]->ptr = children[i];
new_node->children[i]->children = NULL;
00058
00059
00060
00061
        new_node->next = NULL;
00063
        return new_node;
00064 }
```

7.13.1.2 free_node()

```
void free_node (
              struct Node * current )
Definition at line 93 of file linked_list.c.
00094 {
        if (current->size > 0)
00095
00096
00097
            for (size_t i = 0; i < current->size; ++i)
00098
              {
00099
                free (current->children[i]->ptr);
00100
                free (current->children[i]);
00101
            free (current->children);
00102
          }
00103
00104
       free (current->ptr);
00105 free (current);
00106 }
```

7.13.1.3 list_delete_node()

Definition at line 109 of file linked list.c.

```
00110 {
       pthread_mutex_lock (&list_mutex);
00111
       struct Node *current = *head;
00113
       struct Node *prev = NULL;
00114
00115
       if (current && current->ptr == ptr)
00116
00117
            *head = current->next;
00118
            free_node (current);
00119
           pthread_mutex_unlock (&list_mutex);
00120
           return;
00121
00122
00123
       while (current && current->ptr != ptr)
00124
        {
00125
         prev = current;
00126
           current = current->next;
00127
         }
00128
00129
       if (!current)
00130
00131
           pthread_mutex_unlock (&list_mutex);
00132
00133
00134
       prev->next = current->next;
00135
       free_node (current);
00136
       pthread_mutex_unlock (&list_mutex);
00138 }
```

7.13.1.4 list_free()

Definition at line 141 of file linked list.c.

```
00142 {
00143  pthread_mutex_lock (&list_mutex);
```

```
struct Node *current = *head;
00145
        struct Node *next;
00146
00147
        while (current)
00148
        {
00149
            next = current->next;
00150
            free_node (current);
00151
            current = next;
00152
00153
       *head = NULL;
00154
vilead = NoLL;
00155 pthread_mutex_unlock (&list_mutex);
00156 }
```

7.13.1.5 list insert()

Definition at line 68 of file linked list.c.

```
00069 {
00070
        struct Node *new_node = create_node (ptr, amount, children);
00071
        if (!new_node)
00072
00073
            return ERROR_CODE;
00074
00075
       pthread_mutex_lock (&list_mutex);
00076
        if (!*head)
        {
00077
         *head = new_node;
00078
00079
            pthread_mutex_unlock (&list_mutex);
        pthread_mutex_unlock
  return SUCCESS_CODE;
}
08000
00081
00082
        struct Node *temp = *head;
00083
        while (temp->next)
       {
00084
00085
            temp = temp->next;
00086
       temp->next = new_node;
pthread_mutex_unlock (&list_mutex);
00087
00088
00089
        return SUCCESS_CODE;
00090 }
```

7.13.2 Variable Documentation

7.13.2.1 list mutex

```
pthread_mutex_t list_mutex = PTHREAD_MUTEX_INITIALIZER
```

Definition at line 15 of file linked_list.c.

7.14 linked_list.c

```
Go to the documentation of this file.

00001 #include "linked_list.h"

00002 #include "config.h"

00003

00004 #include <pthread.h>

00005 #include <stdlib.h>
```

struct Node **children;

00015 pthread_mutex_t list_mutex = PTHREAD_MUTEX_INITIALIZER;

00019 create_node (void *ptr, size_t amount, void **children)

struct Node *new_node = (struct Node *)malloc (sizeof (struct Node));

= (struct Node **) malloc (sizeof (struct Node *) * amount);

new_node->children[i] = (struct Node *)malloc (sizeof (struct Node));

struct Node *next;

if (!new_node)

return NULL;

new_node->ptr = ptr; new_node->size = amount;

if (amount <= 0)</pre>

new_node->children

new_node->children = NULL;

return new_node;

if (!new_node->children)

free (new_node);

for (size_t i = 0; i < amount; ++i)</pre>

if (!new_node->children[i])

free (new_node);

for (size_t i = 0; i < amount; ++i)</pre>

new_node->children[i]->size = 0;
new_node->children[i]->next = NULL;

new_node->children[i]->ptr = children[i];

00068 list_insert (struct Node **head, void *ptr, size_t amount, void **children)

struct Node *new_node = create_node (ptr, amount, children);

new_node->children[i]->children = NULL;

return NULL;

new_node->next = NULL;

return ERROR_CODE;

*head = new node;

struct Node *temp = *head;

return SUCCESS_CODE;

pthread_mutex_lock (&list_mutex);

pthread_mutex_unlock (&list_mutex);

return new_node;

if (!new_node)

if (!*head)

for (size_t j = 0; j < i; ++j)</pre>

free (new_node->children[j]);

return NULL:

new_node->next = NULL;

00006

} 80000

00009

00011

00012

00016

00020 {

00021

00024

00025

00027 00028

00030

00031

00032

00033

00035

00036

00037

00038

00039

00040 00041

00042 00043

00044

00046 00047

00048

00050

00052

00053 00054 00055

00056

00058

00059

00061 00062

00063

00064 }

00069 {

00071

00072

00074

00076

00077 00078

08000

00081 00082

00066 [[nodiscard]]

00013 }; 00014

00007 struct Node

00010 size_t size;

00017 [[nodiscard]] 00018 struct Node *

{

{

{

void *ptr;

```
while (temp->next)
00084
        {
00085
           temp = temp->next;
00086
       temp->next = new_node;
00087
00088
       pthread_mutex_unlock (&list_mutex);
        return SUCCESS_CODE;
00090 }
00091
00092 void
00093 free_node (struct Node *current)
00094 {
00095
        if (current->size > 0)
00096
00097
            for (size_t i = 0; i < current->size; ++i)
00098
                free (current->children[i]->ptr);
00099
00100
               free (current->children[i]);
00102
           free (current->children);
00103
00104 free (current->ptr);
00105 free (current);
00106 }
00107
00108 void
00109 list_delete_node (struct Node **head, const void *const ptr)
00110 {
00111
       pthread_mutex_lock (&list_mutex);
       struct Node *current = *head;
struct Node *prev = NULL;
00112
00113
00114
00115
       if (current && current->ptr == ptr)
00116
00117
           *head = current->next;
00118
           free_node (current);
          pthread_mutex_unlock (&list_mutex);
00119
           return;
00121
00122
00123
       while (current && current->ptr != ptr)
       {
00124
         prev = current;
00125
00126
           current = current->next;
00127
00128
00129
       if (!current)
00130
          pthread_mutex_unlock (&list_mutex);
00131
00132
            return:
00133
00134
00135
       prev->next = current->next;
00136
       free_node (current);
       pthread_mutex_unlock (&list_mutex);
00137
00138 }
00140 void
00141 list_free (struct Node **head)
00142 {
00143
       pthread mutex lock (&list mutex);
00144
       struct Node *current = *head;
struct Node *next;
00145
00146
00147
       while (current)
00148
00149
           next = current->next;
            free_node (current);
00150
           current = next;
00151
         }
00152
00153
00154
       *head = NULL;
00155
       pthread_mutex_unlock (&list_mutex);
00156 }
```

7.15 / 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
- · struct chunk params

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_fd (struct saurion *const s, int client_socket, int sel)
- 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)
- static size_t calculate_max_iov_content (const struct request *req)
- static int handle_previous_message (struct chunk_params *p)
- static int handle_partial_message (struct chunk_params *p)
- static int handle new message (struct chunk params *p)
- static int prepare destination (struct chunk params *p)
- static void copy_data (struct chunk_params *p, uint8_t *ok)
- static uint8_t validate_and_update (struct chunk_params *p, uint8_t ok)
- static void read_chunk_free (struct chunk_params *p)
- int read chunk (void **dest, size t *len, struct request *const reg)

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 handle_write (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 saurion_set_socket (const int p)

Creates a socket.

• struct saurion * saurion_create (uint32_t n_threads)

Creates an instance of the saurion structure.

- static void handle_event_read (const struct io_uring_cqe *const cqe, struct saurion *const s, struct request *req)
- 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

• static struct timespec TIMEOUT RETRY SPEC = { 0, TIMEOUT RETRY * 1000L }

7.15.1 Macro Definition Documentation

7.15.1.1 EV_ACC

#define EV_ACC 0

Definition at line 26 of file low saurion.c.

7.15.1.2 EV_ERR

```
#define EV_ERR 4
```

Definition at line 30 of file low_saurion.c.

7.15.1.3 EV_REA

```
#define EV_REA 1
```

Definition at line 27 of file low_saurion.c.

7.15.1.4 EV_WAI

```
#define EV_WAI 3
```

Definition at line 29 of file low_saurion.c.

7.15.1.5 EV_WRI

```
#define EV_WRI 2
```

Definition at line 28 of file low_saurion.c.

7.15.1.6 MAX

```
#define MAX(  a, \\ b ) \ ((a) > (b) \ ? \ (a) : \ (b))
```

Definition at line 46 of file low_saurion.c.

7.15.1.7 MIN

Definition at line 45 of file low_saurion.c.

7.15.2 Function Documentation

7.15.2.1 add accept()

```
static void add_accept (
               struct saurion *const s,
               struct sockaddr_in *const ca,
               socklen_t *const cal ) [inline], [static]
Definition at line 257 of file low saurion.c.
00259 {
00260
        int res = ERROR CODE;
        pthread_mutex_lock (&s->m_rings[0]);
while (res != SUCCESS_CODE)
00261
00262
00263
00264
            struct io_uring_sqe *sqe = io_uring_get_sqe (&s->rings[0]);
00265
            while (!sqe)
00266
              {
00267
                sqe = io_uring_get_sqe (&s->rings[0]);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00268
00269
00270
            struct request *req = NULL;
00271
            if (!set_request (&req, &s->list, 0, NULL, 0))
00272
00273
                free (sqe);
00274
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00275
                 res = ERROR_CODE;
00276
                continue;
              1
00277
            req->client_socket = 0;
00278
00279
            req->event_type = EV_ACC;
            io_uring_prep_accept (sqe, s->ss, (struct sockaddr *const)ca, cal, 0);
00280
00281
            io_uring_sqe_set_data (sqe, req);
00282
            if (io_uring_submit (&s->rings[0]) < 0)</pre>
00283
00284
                free (sqe);
                 list_delete_node (&s->list, req);
00285
                 nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00286
00287
                res = ERROR_CODE;
00288
                 continue;
00289
00290
            res = SUCCESS_CODE;
00291
       pthread_mutex_unlock (&s->m_rings[0]);
00292
00293 }
```

7.15.2.2 add efd()

Definition at line 336 of file low_saurion.c.

```
00337 {
00338    add_fd (s, client_socket, sel);
00339 }
```

7.15.2.3 add_fd()

```
static void add fd (
               struct saurion *const s,
               int client_socket,
               int sel ) [inline], [static]
Definition at line 297 of file low saurion.c.
00298 {
00299
        int res = ERROR_CODE;
00300
        pthread_mutex_lock (&s->m_rings[sel]);
        while (res != SUCCESS_CODE)
00301
00302
            struct io_uring *ring = &s->rings[sel];
00304
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00305
            while (!sqe)
00306
                sqe = io_uring_get_sqe (ring);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00307
00308
00309
00310
            struct request *req = NULL;
             if (!set_request (&req, &s->list, CHUNK_SZ, NULL, 0))
00311
00312
00313
                free (sqe);
res = ERROR_CODE;
00314
00315
                continue;
00316
00317
            req->event_type = EV_REA;
00318
             req->client_socket = client_socket;
            io_uring_prep_readv (sqe, client_socket, &req->iov[0], req->iovec_count,
00319
00320
                                   0);
            io_uring_sqe_set_data (sqe, req);
00321
00322
               (io_uring_submit (ring) < 0)
00323
00324
                 free (sqe);
                list_delete_node (&s->list, req);
res = ERROR_CODE;
00325
00326
00327
                continue;
00329
            res = SUCCESS_CODE;
00330
       pthread_mutex_unlock (&s->m_rings[sel]);
00331
```

7.15.2.4 add_read()

00332 }

Definition at line 343 of file low_saurion.c.

```
00344 {
00345    int sel = next (s);
00346    add_fd (s, client_socket, sel);
00347 }
```

7.15.2.5 add_read_continue()

```
Definition at line 351 of file low_saurion.c.
```

```
00354
        pthread_mutex_lock (&s->m_rings[sel]);
        int res = ERROR_CODE;
00355
        while (res != SUCCESS_CODE)
00356
00357
          {
00358
            struct io_uring *ring = &s->rings[sel];
00359
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00360
            while (!sqe)
00361
                sqe = io_uring_get_sqe (ring);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00362
00363
00364
00365
               (!set_request (&oreq, &s->list, oreq->prev_remain, NULL, 0))
00366
              {
00367
                 free (sqe);
                 res = ERROR_CODE;
00368
                continue;
00369
00370
00371
            io_uring_prep_readv (sqe, oreq->client_socket, &oreq->iov[0],
00372
                                   oreq->iovec_count, 0);
00373
            io_uring_sqe_set_data (sqe, oreq);
00374
            if (io_uring_submit (ring) < 0)</pre>
00375
00376
                 free (sqe);
00377
                 list_delete_node (&s->list, oreq);
00378
                 res = ERROR_CODE;
00379
                continue;
00380
00381
            res = SUCCESS CODE;
00382
00383
       pthread_mutex_unlock (&s->m_rings[sel]);
00384 }
```

7.15.2.6 add_write()

Definition at line 388 of file low saurion.c.

```
00390 {
00391
         int res = ERROR_CODE;
00392
         pthread_mutex_lock (&s->m_rings[sel]);
00393
         while (res != SUCCESS_CODE)
00394
             struct io_uring *ring = &s->rings[sel];
struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00395
00396
00397
             while (!sqe)
00398
               {
00399
                  sqe = io_uring_get_sqe (ring);
                  nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00400
00401
00402
             struct request *req = NULL;
             if (!set_request (&req, &s->list, strlen (str), (const void *const)str,
00403
00404
                                  1))
00405
00406
                 free (sqe);
00407
                  res = ERROR_CODE;
00408
                  continue;
00409
                }
             req->event_type = EV_WRI;
req->client_socket = fd;
00410
00411
00412
             io_uring_prep_writev (sqe, req->client_socket, req->iov,
00413
                                       req->iovec_count, 0);
             io_uring_sqe_set_data (sqe, req);
if (io_uring_submit (ring) < 0)</pre>
00414
00415
00416
               {
00417
                  free (sqe);
00418
                  list_delete_node (&s->list, req);
00419
                  res = ERROR_CODE;
                  nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00420
00421
                  continue:
00422
00423
             res = SUCCESS_CODE;
```

```
00424     }
00425     pthread_mutex_unlock (&s->m_rings[sel]);
00426 }
```

7.15.2.7 calculate_max_iov_content()

7.15.2.8 copy_data()

Definition at line 584 of file low_saurion.c.

```
00585 {
00586
       size_t curr_iov_msg_rem = 0;
       *ok = 1UL;
00587
00588
       while (1)
       {
00590
           curr_iov_msg_rem = MIN (
00591
              p->cont_rem, (p->req->iov[p->curr_iov].iov_len - p->curr_iov_off));
           00592
00593
00594
          curr_iov_msg_rem);
p->dest_off += curr_iov_msg_rem;
00595
00596
           p->curr_iov_off += curr_iov_msg_rem;
00597
           p->cont_rem -= curr_iov_msg_rem;
00598
00599
           if (p->cont_rem <= 0)</pre>
00600
00601
               if (*(((uint8_t *)p->req->iov[p->curr_iov].iov_base)
00602
                    + p->curr_iov_off)
                   != 0)
00603
00604
                  *ok = 0UL;
00605
                }
00606
               *p->len = p->cont_sz;
00607
00608
               ++p->curr_iov_off;
00609
00610
00611
           if (p->curr_iov_off >= (p->req->iov[p->curr_iov].iov_len))
00612
00613
               ++p->curr_iov;
00614
               if (p->curr_iov == p->req->iovec_count)
00615
               {
00616
                  break;
00617
00618
              p->curr_iov_off = 0;
00619
00620
         }
00621 }
```

7.15.2.9 handle_accept()

```
static void handle_accept (  {\tt const \ struct \ saurion \ *const \ s,}   {\tt const \ int \ } fd \ ) \quad [inline], \ [static]
```

Definition at line 431 of file low_saurion.c.

7.15.2.10 handle_close()

Definition at line 785 of file low_saurion.c.

7.15.2.11 handle_error()

```
static void handle_error (  {\it const struct saurion *const s,} \\ {\it const struct request *const req ) } \ [inline], \ [static]
```

Definition at line 773 of file low saurion.c.

7.15.2.12 handle_event_read()

```
static void handle_event_read (
             const struct io_uring_cqe *const cqe,
              struct saurion *const s,
              struct request * req ) [inline], [static]
Definition at line 951 of file low saurion.c.
00954
        if (cqe->res < 0)
00955
00956
           handle_error (s, req);
00957
       if (cqe->res < 1)
00959
00960
           handle_close (s, req);
00961
00962
       if (cqe->res > 0)
00963
       {
00964
           handle_read (s, req);
00965
00966
       list_delete_node (&s->list, req);
00967 }
```

7.15.2.13 handle_new_message()

Definition at line 532 of file low_saurion.c.

```
00534
        p->curr_iov = 0;
       p->curr_iov_off = 0;
00535
00536
00537
       p->cont_sz = *(size_t *)((uint8_t *)p->req->iov[p->curr_iov].iov_base
00538
                                 + p->curr_iov_off);
00539 p->cont_sz = ntohll (p->cont_sz);
00540
       p->curr_iov_off += sizeof (uint64_t);
        p->cont_rem = p->cont_sz;
p->dest_off = p->cont_sz - p->cont_rem;
00541
00542
00543
00544
        if (p->cont_rem <= p->max_iov_cont)
00545
00546
            *p->dest = malloc (p->cont_sz);
00547
            if (!*p->dest)
00548
             {
                return ERROR_CODE; // Error al asignar memoria.
00549
00550
00551
            p->dest_ptr = *p->dest;
00552
00553
00554
00555
           p->req->prev = malloc (p->cont_sz);
00556
            if (!p->req->prev)
00557
            {
00558
               return ERROR_CODE; // Error al asignar memoria.
00559
00560
            p->dest_ptr = p->req->prev;
00561
            *p->dest = NULL;
00562
       return SUCCESS_CODE;
00563
00564 }
```

7.15.2.14 handle_partial_message()

```
static int handle_partial_message (
              struct chunk_params * p ) [inline], [static]
Definition at line 494 of file low saurion.c.
00495
00496
        p->curr_iov = p->req->next_iov;
       p->curr_iov_off = p->req->next_offset;
00497
00498
00499
       p->cont_sz = *(size_t *)((uint8_t *)p->req->iov[p->curr_iov].iov_base
00500
                                 + p->curr_iov_off);
00501
00502
       p->cont_sz = ntohll (p->cont_sz);
       p->curr_iov_off += sizeof (uint64_t);
p->cont_rem = p->cont_sz;
00504
       p->dest_off = p->cont_sz - p->cont_rem;
00505
00506
       if ((p->curr_iov_off + p->cont_rem + 1) <= p->max_iov_cont)
00507
00508
            *p->dest = malloc (p->cont_sz);
00509
           if (!*p->dest)
           {
00510
00511
               return ERROR_CODE;
00512
00513
           p->dest_ptr = *p->dest;
00514
         }
00515
       else
00516
       {
00517
           p->req->prev = malloc (p->cont_sz);
         if (!p->req->prev)
00518
00519
00520
               return ERROR_CODE;
00521
          p->dest_ptr = p->req->prev;
00523
           *p->dest = NULL;
00524
            *p->len = 0;
00525
      return SUCCESS_CODE;
00526
00527 }
```

7.15.2.15 handle previous message()

Definition at line 469 of file low saurion.c.

```
00470 {
          p->cont_sz = p->req->prev_size;
00471
          p->cont_sz - p->req->prev_size;
p->cont_rem = p->req->prev_remain;
p->dest_off = p->cont_sz - p->cont_rem;
00472
00473
00474
00475
          if (p->cont_rem <= p->max_iov_cont)
00476
              *p->dest = p->req->prev;
p->dest_ptr = *p->dest;
p->req->prev = NULL;
p->req->prev_size = 0;
00477
00478
00479
00480
00481
                p->req->prev_remain = 0;
00482
00483
          else
           p->dest_ptr = p->req->prev;
*p->dest = NULL;
}
00484
          {
00485
00487
00488
          return SUCCESS_CODE;
00489 }
```

7.15.2.16 handle_read()

```
static void handle_read (
              struct saurion *const s,
              struct request *const req ) [inline], [static]
Definition at line 724 of file low_saurion.c.
00725 {
       void *msg = NULL;
00726
       size_t len = 0;
while (1)
00727
00728
00729
         {
00730
            if (!read_chunk (&msg, &len, req))
00731
               break:
00732
00733
00734
           if (req->next_iov || req->next_offset)
00735
00736
               if (s->cb.on_readed && msg)
00737
00738
                   s->cb.on_readed (req->client_socket, msg, len,
00739
                                    s->cb.on_readed_arg);
00740
00741
               free (msg);
00742
               msg = NULL;
00743
               continue;
00744
00745
           if (req->prev && req->prev_size && req->prev_remain)
00746
            {
00747
               add_read_continue (s, req, next (s));
00748
               return;
00749
00750
            if (s->cb.on_readed && msg)
00751
            {
00752
               s->cb.on_readed (req->client_socket, msg, len, s->cb.on_readed_arg);
00753
00754
           free (msg);
00755
           msg = NULL;
00756
           break;
00757
00758
       add_read (s, req->client_socket);
```

7.15.2.17 handle_write()

00759 }

Definition at line 763 of file low_saurion.c.

7.15.2.18 htonll()

Definition at line 66 of file low_saurion.c.

```
00067 {
00068 int num = 42;
```

7.15.2.19 next()

```
static uint32_t next ( struct \ saurion \ * \ s \ ) \ \ [inline], \ [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 }
```

7.15.2.20 ntohll()

Definition at line 80 of file low_saurion.c.

```
00081 {
00082
         int num = 42;
00083
         if (*(char *)&num == 42)
00084
          {
              uint32_t high_part = ntohl ((uint32_t) (value » 32));
uint32_t low_part = ntohl ((uint32_t) (value & 0xFFFFFFFLL));
00085
00086
00087
              return ((uint64_t)low_part « 32) | high_part;
00088
00089
         return value;
00090 }
```

7.15.2.21 prepare_destination()

Definition at line 569 of file low saurion.c.

7.15.2.22 read_chunk_free()

```
static void read chunk free (
              struct chunk_params * p ) [inline], [static]
Definition at line 662 of file low saurion.c.
00663
00664
        free (p->dest_ptr);
        p->dest_ptr = NULL;
00665
00666
        *p->dest = NULL;
        *p->len = 0;
00667
00668
        p->req->next_iov = 0;
        p->req->next_offset = 0;
for (size_t i = p->curr_iov; i < p->req->iovec_count; ++i)
00669
00670
00672
            for (size_t j = p->curr_iov_off; j < p->req->iov[i].iov_len; ++j)
00673
00674
                uint8_t foot = *((uint8_t *)p->req->iov[i].iov_base) + j;
00675
                if (foot == 0)
00676
                  {
00677
                    p->req->next_iov = i;
00678
                    p->req->next_offset = (j + 1) % p->req->iov[i].iov_len;
00679
                    return;
00680
00681
              }
00682
          }
00683 }
```

7.15.2.23 saurion worker master()

Definition at line 1029 of file low_saurion.c.

```
01030 {
        LOG_INIT (" ");
01031
        struct saurion *const s = (struct saurion *)arg;
struct sockaddr_in client_addr;
01032
01033
01034
        socklen_t client_addr_len = sizeof (client_addr);
01035
01036
        add_efd (s, s->efds[0], 0);
01037
        add_accept (s, &client_addr, &client_addr_len);
01038
01039
        pthread_mutex_lock (&s->status_m);
01040
        ++s->status;
01041
        pthread_cond_broadcast (&s->status_c);
01042
        pthread_mutex_unlock (&s->status_m);
01043
        while (1)
01044
         {
01045
01046
                 = saurion_worker_master_loop_it (s, &client_addr, &client_addr_len);
01047
             if (ret == ERROR_CODE || ret == CRITICAL_CODE)
01048
              {
01049
                break;
01050
              }
01051
01052
       pthread_mutex_lock (&s->status_m);
01053
01054
        pthread_cond_signal (&s->status_c);
        pthread_mutex_unlock (&s->status_m);
LOG_END (" ");
01055
01056
01057
        return;
01058 }
```

7.15.2.24 saurion_worker_master_loop_it()

```
static int saurion_worker_master_loop_it (
               struct saurion *const s,
               struct sockaddr_in * client_addr,
               socklen_t * client_addr_len ) [inline], [static]
Definition at line 972 of file low saurion.c.
        LOG_INIT (" ");
00976
00977
        struct io_uring ring = s->rings[0];
00978
        struct io_uring_cqe *cqe = NULL;
00979
        int ret = io_uring_wait_cqe (&ring, &cqe);
        if (ret < 0)
00980
00981
        {
            free (cqe);
LOG_END (" ");
00982
00983
            return CRITICAL_CODE;
00984
00985
00986
       struct request *req = (struct request *)cqe->user_data;
00987
        if (!req)
00988
         {
00989
            io_uring_cqe_seen (&s->rings[0], cqe);
LOG_END (" ");
00990
            return SUCCESS_CODE;
00991
00992
00993
        if (cqe->res < 0)
00994
            list_delete_node (&s->list, req);
LOG_END (" ");
00995
00996
00997
            return CRITICAL_CODE;
00998
00999
        if (req->client_socket == s->efds[0])
01000
        {
01001
            io_uring_cqe_seen (&s->rings[0], cqe);
            list_delete_node (&s->list, req);
LOG_END (" ");
01002
01003
01004
            return ERROR_CODE;
01005
01006
       io uring cge seen (&s->rings[0], cge);
01007
       switch (req->event_type)
01008
01009
          case EV_ACC:
          handle_accept (s, cqe->res);
add_accept (s, client_addr, client_addr_len);
01010
01011
            add_read (s, cqe->res);
01012
01013
            list_delete_node (&s->list, req);
01014
            break;
01015
         case EV_REA:
01016
           handle_event_read (cqe, s, req);
01017
           break:
          case EV_WRI:
01018
01019
           handle_write (s, req->client_socket);
01020
            list_delete_node (&s->list, req);
01021
            break;
01022
       LOG_END (" ");
01023
01024
       return SUCCESS_CODE;
01025 }
```

7.15.2.25 saurion_worker_slave()

```
void saurion_worker_slave (
               void * arg )
Definition at line 1114 of file low saurion.c.
01115 {
01116
        LOG_INIT (" ");
01117
        struct saurion_wrapper *const ss = (struct saurion_wrapper *)arg;
        struct saurion *s = ss->s;
const int sel = ss->sel;
01118
01119
01120
        free (ss);
01121
01122
        add_efd (s, s->efds[sel], sel);
```

```
01123
01124
        pthread_mutex_lock (&s->status_m);
         ++s->status;
01125
        pthread_cond_broadcast (&s->status_c);
01126
        pthread_mutex_unlock (&s->status_m);
01127
01128
        while (1)
01129
01130
            int res = saurion_worker_slave_loop_it (s, sel);
01131
            if (res == ERROR_CODE || res == CRITICAL_CODE)
01132
                break:
01133
              }
01134
01135
01136
       pthread_mutex_lock (&s->status_m);
01137
         --s->status;
01138
        pthread_cond_signal (&s->status_c);
       pthread_mutex_unlock (&s->status_m);
LOG_END (" ");
01139
01140
01141
        return;
01142 }
```

7.15.2.26 saurion worker slave loop it()

Definition at line 1063 of file low_saurion.c.

```
01065
        LOG_INIT (" ");
        struct io_uring ring = s->rings[sel];
01066
        struct io_uring_cqe *cqe = NULL;
01067
01068
01069
        add_efd (s, s->efds[sel], sel);
01070
        int ret = io_uring_wait_cqe (&ring, &cqe);
01071
        if (ret < 0)
01072
         {
01073
            free (cqe);
LOG_END (" ");
01074
01075
            return CRITICAL_CODE;
01076
01077
        struct request *req = (struct request *)cqe->user_data;
01078
        if (!req)
01079
            io_uring_cqe_seen (&ring, cqe);
LOG_END (" ");
01080
01081
            return SUCCESS_CODE;
01082
01083
01084
        if (cqe->res < 0)</pre>
01085
         {
            list_delete_node (&s->list, req);
LOG_END (" ");
01086
01087
            return CRITICAL_CODE;
01089
01090
        if (req->client_socket == s->efds[sel])
01091
01092
            io_uring_cqe_seen (&ring, cqe);
            list_delete_node (&s->list, req);
LOG_END (" ");
01093
01094
01095
            return ERROR_CODE;
01096
01097
        io_uring_cqe_seen (&ring, cqe);
01098
        switch (req->event_type)
01099
         {
case EV_REA:
01100
           handle_event_read (cqe, s, req);
01101
01102
            break;
01103
           case EV_WRI:
            handle_write (s, req->client_socket);
01104
01105
             list_delete_node (&s->list, req);
01106
            break;
01107
01108
        LOG_END (" ");
        return SUCCESS_CODE;
01109
01110 }
```

7.16 low_saurion.c 89

7.15.2.27 validate_and_update()

```
static uint8_t validate_and_update (
               struct chunk_params * p,
                uint8_t ok ) [inline], [static]
Definition at line 626 of file low_saurion.c.
00628
        if (p->req->prev)
00629
            p->req->prev_size = p->cont_sz;
p->req->prev_remain = p->cont_rem;
00630
00631
00632
            *p->dest = NULL;
            \star p \rightarrow len = 0;
00633
00634
00635
        else
00636
        {
00637
            p->req->prev size = 0;
            p->req->prev_remain = 0;
00639
00640
        if (p->curr_iov < p->req->iovec_count)
00641
        {
             uint64_t next_sz
00642
                 = *(uint64_t *)(((uint8_t *)p->req->iov[p->curr_iov].iov_base)
+ p->curr_iov_off);
00643
00644
             if ((p->req->iov[p->curr_iov].iov_len > p->curr_iov_off) && next_sz)
00646
                 p->req->next_iov = p->curr_iov;
p->req->next_offset = p->curr_iov_off;
00647
00648
00649
00650
            else
00651
             {
00652
                p->req->next_iov = 0;
00653
                p->req->next_offset = 0;
00654
00655
        }
00656
        return ok ? SUCCESS_CODE : ERROR_CODE;
```

7.15.3 Variable Documentation

7.15.3.1 TIMEOUT_RETRY_SPEC

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

Definition at line 48 of file low_saurion.c.

7.16 low_saurion.c

Go to the documentation of this file.

```
00015 #include <stdlib.h>
                                       // for free, malloc
                                       // for memset, memcpy, strlen
// for eventfd, EFD_NONBLOCK
00016 #include <string.h>
00017 #include <sys/eventfd.h>
                                       // for socklen_t, bind, listen, setsockopt
00018 #include <sys/socket.h>
                                       // for iovec
00019 #include <sys/uio.h>
                                       // for nanosleep
00020 #include <time.h>
00021 #include <unistd.h>
                                       // for close, write
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 3
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
00048 static struct timespec TIMEOUT_RETRY_SPEC = { 0, TIMEOUT_RETRY * 1000L };
00049
00050 struct saurion_wrapper
00051 {
       struct saurion *s;
00053
        uint32_t sel;
00054 };
00055
00056 // next
00057 static inline uint32 t
00058 next (struct saurion *s)
00059 {
00060 s->next = (s->next + 1) % s->n_threads;
00061 return s->next;
00062 }
00063
00064 // htonll
00065 static inline uint64_t
00066 htonll (uint64_t value)
00067 {
00068
        int num = 42;
00069
        if (*(char *)&num == 42)
        {
00070
00071
            uint32_t high_part = hton1 ((uint32_t) (value » 32));
00072
            uint32_t low_part = hton1 ((uint32_t) (value & 0xFFFFFFFFLL));
           return ((uint64_t)low_part « 32) | high_part;
00073
00074
00075
       return value;
00076 }
00077
00078 // ntohll
00079 static inline uint64_t
00080 ntohll (uint64_t value)
00081 {
00082
        int num = 42;
00083
        if (*(char *)&num == 42)
        {
            uint32_t high_part = ntohl ((uint32_t)(value » 32));
uint32_t low_part = ntohl ((uint32_t)(value & 0xFFFFFFFLL));
00085
00086
00087
           return ((uint64_t)low_part « 32) | high_part;
00088
00089
       return value;
00090 }
00091
00092 // free_request
00093 void
00094 free_request (struct request *req, void **children_ptr, size_t amount)
00095 {
        if (children_ptr)
00097
00098
            free (children_ptr);
00099
            children_ptr = NULL;
00100
00101
       for (size_t i = 0; i < amount; ++i)</pre>
```

7.16 low_saurion.c 91

```
{
00103
             free (req->iov[i].iov_base);
00104
            req->iov[i].iov_base = NULL;
00105
00106
        free (reg);
        req = NULL;
00107
        free (children_ptr);
00109
        children_ptr = NULL;
00110 }
00111
00112 // initialize_iovec
00113 [[nodiscard]]
00114 int
00115 initialize_iovec (struct iovec *iov, size_t amount, size_t pos,
00116
                         const void *msg, size_t size, uint8_t h)
00117 {
        if (!iov || !iov->iov_base)
00118
00119
        {
            return ERROR_CODE;
00121
00122
        if (msg)
00123
            size_t len = iov->iov_len;
char *dest = (char *)iov->iov_base;
char *orig = (char *)msg + pos * CHUNK_SZ;
00124
00125
00126
00127
            size_t cpy_sz = 0;
00128
             if (h)
00129
                 if (pos == 0)
00130
00131
                   {
                    uint64_t send_size = htonll (size);
memcpy (dest, &send_size, sizeof (uint64_t));
dest += sizeof (uint64_t);
00132
00133
00134
00135
                     len -= sizeof (uint64_t);
00136
00137
                 else
00138
                  {
00139
                     orig -= sizeof (uint64_t);
00140
00141
                 if ((pos + 1) == amount)
00142
                     --len;
00143
                     cpy_sz = (len < size ? len : size);
00144
00145
                     dest[cpy_sz] = 0;
00146
00147
              }
00148
            cpy_sz = (len < size ? len : size);
00149
            memcpy (dest, orig, cpy_sz);
            dest += cpy_sz;
size_t rem = CHUNK_SZ - (dest - (char *)iov->iov_base);
00150
00151
00152
            memset (dest, 0, rem);
00153
00154
        else
00155
        {
            memset ((char *)iov->iov_base, 0, CHUNK_SZ);
00156
00157
        return SUCCESS_CODE;
00159 }
00160
00161 // allocate iovec
00162 [[nodiscard]]
00163 int
00164 allocate_iovec (struct iovec *iov, size_t amount, size_t pos, size_t size,
00165
                       void **chd_ptr)
00166 {
00167
        if (!iov || !chd_ptr)
00168
00169
            return ERROR CODE:
00170
        iov->iov_base = malloc (CHUNK_SZ);
00172
        if (!iov->iov_base)
00173
00174
            return ERROR_CODE;
00175
00176
        iov->iov_len = (pos == (amount - 1) ? (size % CHUNK_SZ) : CHUNK_SZ);
00177
        if (iov->iov_len == 0)
00178
        {
00179
            iov->iov_len = CHUNK_SZ;
00180
00181
        chd ptr[pos] = iov->iov base;
        return SUCCESS_CODE;
00182
00183 }
00184
00185 // set_request
00186 [[nodiscard]]
00187 int.
00188 set request (struct request **r, struct Node **l, size t s, const void *m,
```

```
00189
                   uint8_t h)
00190 {
00191
       uint64_t full_size = s;
00192
        if (h)
00193
00194
            full_size += (sizeof (uint64_t) + sizeof (uint8_t));
00195
00196
        size_t amount = full_size / CHUNK_SZ;
00197
        amount = amount + (full_size % CHUNK_SZ == 0 ? 0 : 1);
00198
        struct request *temp = (struct request *)malloc (
            sizeof (struct request) + sizeof (struct iovec) * amount);
00199
00200
        if (!temp)
        {
00201
00202
            return ERROR_CODE;
00203
00204
        if (!*r)
00205
00206
            *r = temp;
            (*r) \rightarrow prev = NULL;
00208
            (*r) \rightarrow prev\_size = 0;
00209
            (*r)->prev_remain = 0;
00210
            (*r) \rightarrow next_iov = 0;
            (*r) \rightarrow next_offset = 0;
00211
00212
00213
       else
00214
        {
00215
            temp->client_socket = (*r)->client_socket;
00216
            temp->event_type = (*r)->event_type;
00217
            temp->prev = (*r)->prev;
            temp->prev_size = (*r)->prev_size;
00218
            temp->prev_remain = (*r)->prev_remain;
00219
00220
            temp->next_iov = (*r)->next_iov;
00221
            temp->next_offset = (*r)->next_offset;
00222
            *r = temp;
00223
00224
        struct request *req = *r;
        req->iovec_count = (int)amount;
00225
        void **children_ptr = (void **)malloc (amount * sizeof (void *));
00227
        if (!children_ptr)
00228
00229
            free_request (req, children_ptr, 0);
            return ERROR_CODE;
00230
00231
00232
        for (size_t i = 0; i < amount; ++i)</pre>
00233
00234
            if (!allocate_iovec (&req->iov[i], amount, i, full_size, children_ptr))
00235
                free_request (req, children_ptr, amount);
00236
00237
                return ERROR CODE;
00238
00239
            if (!initialize_iovec (&req->iov[i], amount, i, m, s, h))
00240
00241
                free_request (req, children_ptr, amount);
00242
                return ERROR_CODE;
00243
00244
00245
        if (!list_insert (1, req, amount, children_ptr))
00246
        {
00247
            free_request (req, children_ptr, amount);
00248
            return ERROR_CODE;
          }
00249
       free (children_ptr);
00250
00251
       return SUCCESS_CODE;
00252 }
00253
00254 /************ ADDERS **************/
00255 // add_accept
00256 static inline void
00257 add_accept (struct saurion *const s, struct sockaddr_in *const ca,
00258
                  socklen_t *const cal)
00259 {
00260
       int res = ERROR_CODE;
00261
        pthread_mutex_lock (&s->m_rings[0]);
00262
        while (res != SUCCESS CODE)
00263
         {
00264
            struct io_uring_sqe *sqe = io_uring_get_sqe (&s->rings[0]);
00265
00266
                sqe = io_uring_get_sqe (&s->rings[0]);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00267
00268
00269
00270
            struct request *req = NULL;
00271
            if (!set_request (&req, &s->list, 0, NULL, 0))
00272
                free (sqe);
00273
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00274
                res = ERROR_CODE;
00275
```

7.16 low_saurion.c 93

```
continue;
00277
00278
            req->client_socket = 0;
00279
            req->event_type = EV_ACC;
00280
            io_uring_prep_accept (sqe, s->ss, (struct sockaddr *const)ca, cal, 0);
00281
            io_uring_sqe_set_data (sqe, reg);
            if (io_uring_submit (&s->rings[0]) < 0)</pre>
00283
00284
                free (sqe);
                list_delete_node (&s->list, req);
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00285
00286
                res = ERROR_CODE;
00287
00288
                continue;
00289
00290
            res = SUCCESS_CODE;
00291
00292
       pthread_mutex_unlock (&s->m_rings[0]);
00293 }
00295 // add_fd
00296 static inline void
00297 add_fd (struct saurion *const s, int client_socket, int sel)
00298 {
00299
        int res = ERROR CODE;
        pthread_mutex_lock (&s->m_rings[sel]);
00300
        while (res != SUCCESS_CODE)
00302
00303
            struct io_uring *ring = &s->rings[sel];
00304
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00305
            while (!sqe)
00306
             {
00307
                sqe = io_uring_get_sqe (ring);
00308
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00309
00310
            struct request *req = NULL;
            if (!set_request (&req, &s->list, CHUNK_SZ, NULL, 0))
00311
00312
             {
                free (sqe);
00313
00314
                res = ERROR_CODE;
00315
                continue;
00316
00317
            req->event_type = EV_REA;
            reg->client socket = client socket;
00318
            io_uring_prep_readv (sqe, client_socket, &req->iov[0], req->iovec_count,
00319
00320
                                 0);
00321
            io_uring_sqe_set_data (sqe, req);
00322
            if (io_uring_submit (ring) < 0)</pre>
00323
00324
                free (sge);
00325
                list_delete_node (&s->list, req);
                res = ERROR_CODE;
00326
00327
                continue;
00328
00329
            res = SUCCESS_CODE;
00330
00331
       pthread_mutex_unlock (&s->m_rings[sel]);
00333
00334 // add_efd
00335 static inline void
00336 add_efd (struct saurion *const s, const int client_socket, int sel)
00337 {
00338
       add_fd (s, client_socket, sel);
00339 }
00340
00341 // add_read
00342 static inline void
00343 add_read (struct saurion *const s, const int client_socket)
00344 {
00345
       int sel = next (s);
00346 add_fd (s, client_socket, sel);
00347 }
00348
00349 // add_read_continue
00350 static inline void
00351 add_read_continue (struct saurion *const s, struct request *oreq,
00352
                         const int sel)
00353 {
00354
       pthread_mutex_lock (&s->m_rings[sel]);
        int res = ERROR_CODE;
00355
00356
        while (res != SUCCESS_CODE)
00357
         {
00358
            struct io_uring *ring = &s->rings[sel];
00359
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00360
            while (!sqe)
00361
              {
00362
                sqe = io_uring_get_sqe (ring);
```

```
nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00364
00365
            if (!set_request (&oreq, &s->list, oreq->prev_remain, NULL, 0))
00366
             {
00367
                free (sge);
                res = ERROR_CODE;
00368
               continue;
00369
00370
00371
            io_uring_prep_readv (sqe, oreq->client_socket, &oreq->iov[0],
00372
                                 oreq->iovec_count, 0);
            io_uring_sqe_set_data (sqe, oreq);
00373
            if (io_uring_submit (ring) < 0)</pre>
00374
00375
             {
00376
                free (sqe);
00377
                list_delete_node (&s->list, oreq);
00378
                res = ERROR_CODE;
00379
               continue:
00380
00381
           res = SUCCESS_CODE;
00382
00383
       pthread_mutex_unlock (&s->m_rings[sel]);
00384 }
00385
00386 // add_write
00387 static inline void
00388 add_write (struct saurion *const s, int fd, const char *const str,
00389
                 const int sel)
00390 {
       int res = ERROR_CODE;
00391
00392
       pthread_mutex_lock (&s->m_rings[sel]);
while (res != SUCCESS_CODE)
00393
00394
         {
00395
           struct io_uring *ring = &s->rings[sel];
00396
            struct io_uring_sqe *sqe = io_uring_get_sqe (ring);
00397
            while (!sqe)
00398
             {
00399
               sqe = io_uring_get_sqe (ring);
               nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00401
00402
            struct request *req = NULL;
            if (!set_request (&req, &s->list, strlen (str), (const void *const)str,
00403
00404
                              1))
00405
             {
00406
               free (sqe);
                res = ERROR_CODE;
00407
00408
                continue;
00409
            req->event_type = EV_WRI;
00410
            req->client_socket = fd;
00411
00412
           io_uring_prep_writev (sqe, req->client_socket, req->iov,
00413
                                 req->iovec_count, 0);
00414
            io_uring_sqe_set_data (sqe, req);
00415
            if (io_uring_submit (ring) < 0)</pre>
00416
             {
               free (sqe);
00417
00418
                list_delete_node (&s->list, req);
                res = ERROR_CODE;
00420
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
00421
00422
           res = SUCCESS CODE:
00423
00424
00425
       pthread_mutex_unlock (&s->m_rings[sel]);
00426 }
00427
00429 // handle_accept
00430 static inline void
00431 handle accept (const struct saurion *const s, const int fd)
00432 {
00433
       if (s->cb.on_connected)
       {
00434
00435
           s->cb.on_connected (fd, s->cb.on_connected_arg);
00436
00437 }
00438
00439 // calculate_max_iov_content
00440 [[nodiscard]]
00441 static inline size_t
00442 calculate max iov content (const struct request *reg)
00443 {
00444
       size_t max_iov_cont = 0;
00445
       for (size_t i = 0; i < req->iovec_count; ++i)
00446
00447
           max_iov_cont += req->iov[i].iov_len;
00448
00449
       return max iov cont;
```

7.16 low_saurion.c 95

```
00450 }
00451
00452 struct chunk_params
00453 {
00454
       void **dest;
00455
       void *dest_ptr;
       size_t dest_off;
00456
00457
       struct request *req;
00458
       size_t cont_sz;
00459
       size_t cont_rem;
00460
       size_t max_iov_cont;
       size_t curr_iov;
00461
       size_t curr_iov_off;
00462
00463
      size_t *len;
00464 };
00465
00466 // handle_previous_message
00467 [[nodiscard]]
00468 static inline int
00469 handle_previous_message (struct chunk_params *p)
00470 {
00471
       p->cont_sz = p->req->prev_size;
        p->cont_rem = p->req->prev_remain;
p->dest_off = p->cont_sz - p->cont_rem;
00472
00473
00474
00475
        if (p->cont_rem <= p->max_iov_cont)
00476
        {
00477
            *p->dest = p->req->prev;
00478
            p->dest_ptr = *p->dest;
            p->req->prev = NULL;
00479
00480
            p->req->prev_size = 0;
00481
            p->req->prev_remain = 0;
00482
00483
        else
00484
        {
         p->dest_ptr = p->req->prev;
*p->dest = NULL;
}
00485
00486
00488
       return SUCCESS_CODE;
00489 }
00490
00491 // handle_partial_message
00492 [[nodiscard]]
00493 static inline int
00494 handle_partial_message (struct chunk_params *p)
00495 {
00496 p->curr_iov = p->req->next_iov;
00497
       p->curr_iov_off = p->req->next_offset;
00498
00499
       p->cont_sz = *(size_t *)((uint8_t *)p->req->iov[p->curr_iov].iov_base
00500
                                  + p->curr_iov_off);
00501
        p->cont_sz = ntohl1 (p->cont_sz);
00502
        p->curr_iov_off += sizeof (uint64_t);
00503
        p->cont_rem = p->cont_sz;
        p->dest_off = p->cont_sz - p->cont_rem;
00504
00505
        if ((p->curr_iov_off + p->cont_rem + 1) <= p->max_iov_cont)
00507
         {
00508
            *p->dest = malloc (p->cont_sz);
00509
            if (!*p->dest)
00510
             {
               return ERROR_CODE;
00511
00512
00513
            p->dest_ptr = *p->dest;
00514
00515
        else
00516
         {
            p->req->prev = malloc (p->cont_sz);
00517
00518
            if (!p->req->prev)
            {
00520
               return ERROR_CODE;
00521
            p->dest_ptr = p->req->prev;
*p->dest = NULL;
00522
00523
            *p->len = 0;
00524
00525
00526
       return SUCCESS_CODE;
00527 }
00528
00529 // handle_new_message
00530 [[nodiscard]]
00531 static inline int
00532 handle_new_message (struct chunk_params *p)
00533 {
00534 p->curr_iov = 0;
       p->curr_iov_off = 0;
00535
00536
```

```
p->cont_sz = *(size_t *)((uint8_t *)p->req->iov[p->curr_iov].iov_base
00538
                                + p->curr_iov_off);
       p->cont_sz = ntohll (p->cont_sz);
00539
00540
       p->curr_iov_off += sizeof (uint64_t);
00541
       p->cont_rem = p->cont_sz;
00542
       p->dest_off = p->cont_sz - p->cont_rem;
00543
00544
       if (p->cont_rem <= p->max_iov_cont)
00545
            *p->dest = malloc (p->cont_sz);
00546
            if (!*p->dest)
00547
00548
             {
00549
               return ERROR_CODE; // Error al asignar memoria.
00550
00551
           p->dest_ptr = *p->dest;
00552
00553
       else
00554
         {
           p->req->prev = malloc (p->cont_sz);
00556
           if (!p->req->prev)
00557
00558
               return ERROR_CODE; // Error al asignar memoria.
00559
           p->dest_ptr = p->req->prev;
00560
00561
           *p->dest = NULL;
00562
00563
       return SUCCESS_CODE;
00564 }
00565
00566 // prepare_destination
00567 [[nodiscard]]
00568 static inline int
00569 prepare_destination (struct chunk_params *p)
00570 {
00571
          (p->req->prev && p->req->prev_size && p->req->prev_remain)
00572
00573
           return handle previous message (p);
00575
          (p->req->next_iov || p->req->next_offset)
00576
00577
           return handle_partial_message (p);
00578
00579
       return handle new message (p);
00580 }
00581
00582 // copy_data
00583 static inline void
00584 copy_data (struct chunk_params *p, uint8_t *ok)
00585 {
00586
       size_t curr_iov_msg_rem = 0;
00587
       *ok = 1UL;
00588
       while (1)
00589
00590
           curr_iov_msg_rem = MIN (
              p->cont_rem, (p->req->iov[p->curr_iov].iov_len - p->curr_iov_off));
00591
           00592
00594
                   curr_iov_msg_rem);
00595
           p->dest_off += curr_iov_msg_rem;
           p->curr_iov_off += curr_iov_msg_rem;
00596
00597
           p->cont_rem -= curr_iov_msg_rem;
00598
00599
            if (p->cont_rem <= 0)</pre>
00600
00601
               if (*(((uint8_t *)p->req->iov[p->curr_iov].iov_base)
                     + p->curr_iov_off)
00602
                   != 0)
00603
00604
                 {
00605
                   *ok = OUL;
00606
00607
               *p->len = p->cont_sz;
00608
               ++p->curr_iov_off;
00609
               break;
00610
              (p->curr_iov_off >= (p->req->iov[p->curr_iov].iov_len))
00611
00612
00613
00614
               if (p->curr_iov == p->req->iovec_count)
00615
00616
                   break:
00617
00618
               p->curr_iov_off = 0;
00619
00620
         }
00621 }
00622
00623 // validate_and_update
```

7.16 low_saurion.c 97

```
00624 [[nodiscard]]
00625 static inline uint8_t
00626 validate_and_update (struct chunk_params *p, uint8_t ok)
00627 {
00628
       if (p->req->prev)
00629
00630
           p->req->prev_size = p->cont_sz;
00631
            p->req->prev_remain = p->cont_rem;
00632
            *p->dest = NULL;
            *p->len = 0;
00633
          }
00634
00635
       else
00636
        {
00637
           p->req->prev_size = 0;
00638
           p->req->prev_remain = 0;
00639
        if (p->curr_iov < p->req->iovec_count)
00640
00641
        {
00642
           uint64_t next_sz
00643
               = *(uint64_t *)(((uint8_t *)p->req->iov[p->curr_iov].iov_base)
00644
                                 + p->curr_iov_off);
00645
            if ((p->req->iov[p->curr_iov].iov_len > p->curr_iov_off) && next_sz)
00646
             {
00647
                p->req->next_iov = p->curr_iov;
00648
                p->req->next_offset = p->curr_iov_off;
00649
00650
            else
00651
             {
00652
                p->req->next_iov = 0;
               p->req->next_offset = 0;
00653
00654
00655
          }
00656
00657
       return ok ? SUCCESS_CODE : ERROR_CODE;
00658 }
00659
00660 // read_chunk_free
00661 static inline void
00662 read_chunk_free (struct chunk_params *p)
00663 {
00664
       free (p->dest_ptr);
00665
       p->dest_ptr = NULL;
00666
       *p->dest = NULL:
       *p->len = 0;
00667
       p->req->next_iov = 0;
00668
00669
        p->req->next_offset = 0;
00670
        for (size_t i = p->curr_iov; i < p->req->iovec_count; ++i)
00671
00672
            for (size_t j = p->curr_iov_off; j < p->req->iov[i].iov_len; ++j)
00673
00674
                uint8_t foot = *((uint8_t *)p->req->iov[i].iov_base) + j;
00675
                <u>if</u> (foot == 0)
00676
00677
                    p->req->next_iov = i;
00678
                    p->req->next_offset = (j + 1) % p->req->iov[i].iov_len;
00679
                    return;
00680
00681
              }
00682
          }
00683 }
00684
00685 // read_chunk
00686 [[nodiscard]]
00687 int
00688 read_chunk (void **dest, size_t *len, struct request *const req)
00689 {
00690
       struct chunk_params p;
00691
       p.req = req;
p.dest = dest;
00692
        p.len = len;
00693
00694
        if (p.req->iovec_count == 0)
00695
00696
            return ERROR_CODE;
00697
00698
00699
       p.max_iov_cont = calculate_max_iov_content (p.req);
00700
        p.cont_sz = 0;
00701
        p.cont_rem = 0;
        p.curr_iov = 0;
00702
00703
        p.curr_iov_off = 0;
00704
        p.dest_off = 0;
p.dest_ptr = NULL;
00705
00706
        if (!prepare_destination (&p))
00707
00708
            return ERROR_CODE;
00709
00710
```

```
00711
       uint8_t ok = 1UL;
00712
       copy_data (&p, &ok);
00713
00714
       if (validate_and_update (&p, ok))
00715
00716
           return SUCCESS_CODE;
00717
00718
       read_chunk_free (&p);
00719
       return ERROR_CODE;
00720 }
00721
00722 // handle read
00723 static inline void
00724 handle_read (struct saurion *const s, struct request *const req)
00725 {
00726
       void *msg = NULL;
00727
       size_t len = 0;
while (1)
00728
00729
        {
00730
           if (!read_chunk (&msg, &len, req))
00731
             {
00732
               break;
00733
00734
           if (req->next_iov || req->next_offset)
00735
             {
00736
               if (s->cb.on_readed && msg)
00737
00738
                   s->cb.on_readed (req->client_socket, msg, len,
00739
                                    s->cb.on_readed_arg);
00740
                 }
00741
               free (msg);
00742
               msg = NULL;
00743
               continue;
00744
00745
           if (req->prev && req->prev_size && req->prev_remain)
00746
00747
               add_read_continue (s, req, next (s));
00748
               return;
00749
00750
           if (s->cb.on_readed && msg)
00751
               s->cb.on_readed (req->client_socket, msg, len, s->cb.on_readed_arg);
00752
00753
00754
           free (msg);
00755
           msg = NULL;
00756
           break;
00757
00758
       add_read (s, req->client_socket);
00759 }
00760
00761 // handle_write
00762 static inline void
00763 handle_write (const struct saurion *const s, const int fd)
00764 {
00765
       if (s->cb.on_wrote)
00766
        {
00767
           s->cb.on_wrote (fd, s->cb.on_wrote_arg);
00768
00769 }
00770
00771 // handle_error
00772 static inline void
00773 handle_error (const struct saurion *const s, const struct request *const req)
00774 {
00775
       if (s->cb.on_error)
00776
       {
           const char *resp = "ERROR";
00777
           s->cb.on_error (req->client_socket, resp, (ssize_t)strlen (resp),
00778
00779
                           s->cb.on error arg);
00780
         }
00781 }
00782
00783 // handle_close
00784 static inline void
00785 handle_close (const struct saurion *const s, const struct request *const req)
00786 {
00787
       if (s->cb.on_closed)
00788
00789
           s->cb.on_closed (req->client_socket, s->cb.on_closed_arg);
00790
00791
       close (req->client_socket);
00792 }
00793
00795 // saurion_set_socket
00796 [[nodiscard]] int
00797 saurion_set_socket (const int p)
```

7.16 low_saurion.c 99

```
00798 {
00799
        int sock = 0;
00800
        struct sockaddr_in srv_addr;
00801
00802
        sock = socket (PF_INET, SOCK_STREAM, 0);
00803
        if (sock < 1)
00805
             return ERROR_CODE;
00806
00807
00808
        int enable = 1:
        if (setsockopt (sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof (int)) < 0)</pre>
00809
00810
         {
00811
             return ERROR_CODE;
00812
00813
        memset (&srv_addr, 0, sizeof (srv_addr));
srv_addr.sin_family = AF_INET;
srv_addr.sin_port = htons (p);
00814
00815
00816
00817
        srv_addr.sin_addr.s_addr = htonl (INADDR_ANY);
00818
00819
        if (bind (sock, (const struct sockaddr \star)&srv_addr, sizeof (srv_addr)) < 0)
00820
         {
            return ERROR CODE:
00821
00822
00824
        if (listen (sock, ACCEPT_QUEUE) < 0)</pre>
00825
00826
            return ERROR_CODE;
00827
00828
00829
        return sock;
00830 }
00831
00832 // saurion_create
00833 [[nodiscard]]
00834 struct saurion
00835 saurion_create (uint32_t n_threads)
00836 {
00837
        LOG_INIT (" ");
00838
        struct saurion *p = (struct saurion *) malloc (size of (struct saurion));
00839
        if (!p)
00840
         {
            LOG_END (" ");
00841
          return NULL;
00842
00843
00844
        int ret = 0;
00845
        ret = pthread_mutex_init (&p->status_m, NULL);
00846
        if (ret)
00847
         {
            free (p);
LOG_END (" ");
00848
00849
00850
             return NULL;
00851
        ret = pthread_cond_init (&p->status_c, NULL);
00852
00853
        if (ret)
00854
             free (p);
LOG_END (" ");
00855
00856
00857
             return NULL;
00858
00859
        p->m_rings
00860
             = (pthread_mutex_t *)malloc (n_threads * sizeof (pthread_mutex_t));
00861
        if (!p->m_rings)
00862
         {
            free (p);
LOG_END (" ");
00863
00864
00865
            return NULL;
00866
00867
        for (uint32_t i = 0; i < n_threads; ++i)</pre>
00868
00869
            pthread_mutex_init (&(p->m_rings[i]), NULL);
00870
00871
        p->ss = 0;
        n_threads = (n_threads < 2 ? 2 : n_threads);
n_threads = (n_threads > NUM_CORES ? NUM_CORES : n_threads);
00872
00873
00874
        p->n_threads = n_threads;
00875
        p->status = 0;
00876
        p->list = NULL;
00877
        p->cb.on_connected = NULL;
00878
        p->cb.on_connected_arg = NULL;
00879
        p->cb.on_readed = NULL;
00880
        p->cb.on_readed_arg = NULL;
00881
        p->cb.on_wrote = NULL;
00882
        p->cb.on_wrote_arg = NULL;
00883
        p->cb.on_closed = NULL;
00884
        p->cb.on_closed_arg = NULL;
```

```
p->cb.on_error = NULL;
00886
        p->cb.on_error_arg = NULL;
00887
        p->next = 0;
        p->efds = (int *)malloc (sizeof (int) * p->n_threads);
00888
00889
        if (!p->efds)
00890
00891
            free (p->m_rings);
            free (p);
LOG_END (" ");
00892
00893
00894
            return NULL;
00895
00896
        for (uint32 t i = 0; i  n threads; ++i)
00897
00898
            p->efds[i] = eventfd (0, EFD_NONBLOCK);
00899
            if (p->efds[i] == ERROR_CODE)
00900
00901
                for (uint32_t j = 0; j < i; ++j)
00902
                 {
00903
                    close (p->efds[j]);
00904
00905
                free (p->efds);
00906
                free (p->m_rings);
00907
                free (p);
LOG_END (" ");
00908
00909
                return NULL;
00910
00911
00912
        p->rings
           = (struct io_uring *)malloc (sizeof (struct io_uring) * p->n_threads);
00913
        if (!p->rings)
00914
00915
00916
            for (uint32_t j = 0; j < p->n_threads; ++j)
00917
00918
                close (p->efds[j]);
00919
00920
            free (p->efds);
00921
            free (p->m_rings);
            free (p);
LOG_END (" ");
00923
00924
            return NULL;
00925
        for (uint32_t i = 0; i < p->n_threads; ++i)
00926
00927
00928
            memset (&p->rings[i], 0, sizeof (struct io_uring));
00929
            ret = io_uring_queue_init (SAURION_RING_SIZE, &p->rings[i], 0);
00930
            if (ret)
00931
                for (uint32_t j = 0; j < p->n_threads; ++j)
00932
00933
                 {
00934
                    close (p->efds[j]);
00935
00936
                free (p->efds);
00937
                free (p->rings);
                free (p->m_rings);
00938
00939
                free (p);
LOG_END (" ");
00940
00941
                return NULL;
00942
00943
       p->pool = threadpool_create (p->n_threads);
LOG_END (" ");
00944
00945
00946
       return p;
00947 }
00948
00949 // handle_event_read
00950 static inline void
00951 handle_event_read (const struct io_uring_cqe *const cqe,
00952
                          struct saurion *const s, struct request *req)
00953 {
00954
        if (cqe->res < 0)</pre>
00955
00956
            handle_error (s, req);
00957
00958
        if (cge->res < 1)</pre>
00959
        {
00960
            handle_close (s, req);
00961
00962
        if (cqe->res > 0)
00963
00964
            handle_read (s, req);
00965
00966
       list_delete_node (&s->list, req);
00967 }
00968
00969 // saurion_worker_master_loop_it
00970 [[nodiscard]]
00971 static inline int
```

7.16 low_saurion.c 101

```
00972 saurion_worker_master_loop_it (struct saurion *const s,
00973
                                        struct sockaddr_in *client_addr,
00974
                                        socklen_t *client_addr_len)
00975 {
        LOG_INIT (" ");
00976
        struct io_uring ring = s->rings[0];
struct io_uring_cqe *cqe = NULL;
00977
00979
        int ret = io_uring_wait_cqe (&ring, &cqe);
00980
        if (ret < 0)
00981
            free (cqe);
LOG_END (" ");
00982
00983
00984
            return CRITICAL_CODE;
00985
00986
        struct request *req = (struct request *)cqe->user_data;
00987
        if (!req)
00988
00989
            io_uring_cqe_seen (&s->rings[0], cqe);
LOG_END (" ");
00990
00991
            return SUCCESS_CODE;
00992
00993
        if (cqe->res < 0)</pre>
00994
         {
            list_delete_node (&s->list, req);
LOG_END (" ");
00995
00996
00997
            return CRITICAL_CODE;
00998
00999
        if (req->client_socket == s->efds[0])
01000
01001
            io_uring_cge_seen (&s->rings[0], cge);
01002
            list_delete_node (&s->list, req);
LOG_END (" ");
01003
01004
            return ERROR_CODE;
01005
01006
        io_uring_cqe_seen (&s->rings[0], cqe);
01007
        switch (req->event_type)
01008
          case EV_ACC:
01010
            handle_accept (s, cqe->res);
01011
             add_accept (s, client_addr, client_addr_len);
01012
             add_read (s, cqe->res);
             list_delete_node (&s->list, req);
01013
01014
            break;
          case EV_REA:
01015
01016
           handle_event_read (cqe, s, req);
01017
            break;
01018
          case EV_WRI:
01019
            handle_write (s, req->client_socket);
01020
             list_delete_node (&s->list, req);
01021
            break:
01022
01023
        LOG_END (" ");
01024
        return SUCCESS_CODE;
01025 }
01026
01027 // saurion_worker_master
01028 void
01029 saurion_worker_master (void *arg)
01030 {
        LOG_INIT (" ");
01031
        struct saurion *const s = (struct saurion *)arg;
01032
        struct sockaddr_in client_addr;
01033
01034
        socklen_t client_addr_len = sizeof (client_addr);
01035
01036
        add_efd (s, s->efds[0], 0);
01037
        add_accept (s, &client_addr, &client_addr_len);
01038
01039
        pthread_mutex_lock (&s->status_m);
01040
        ++s->status;
01041
        pthread_cond_broadcast (&s->status_c);
01042
        pthread_mutex_unlock (&s->status_m);
01043
         while (1)
01044
         {
            int ret
01045
             = saurion_worker_master_loop_it (s, &client_addr, &client_addr_len);
if (ret == ERROR_CODE || ret == CRITICAL_CODE)
01046
01047
01048
              {
01049
                break;
              }
01050
01051
        pthread_mutex_lock (&s->status_m);
01052
01053
         -s->status;
01054
        pthread_cond_signal (&s->status_c);
01055
        pthread_mutex_unlock (&s->status_m);
01056
        LOG_END (" ");
01057
        return;
01058 }
```

```
01060 // saurion_worker_slave_loop_it
01061 [[nodiscard]]
01062 static inline int
01063 saurion_worker_slave_loop_it (struct saurion *const s, const int sel)
01064 {
01066
        struct io_uring ring = s->rings[sel];
01067
        struct io_uring_cqe *cqe = NULL;
01068
01069
        add_efd (s, s->efds[sel], sel);
01070
        int ret = io_uring_wait_cqe (&ring, &cqe);
        if (ret < 0)</pre>
01071
01072
            free (cqe);
LOG_END (" ");
01073
01074
            return CRITICAL_CODE;
01075
01076
01077
        struct request *req = (struct request *)cqe->user_data;
01078
        if (!req)
01079
            io_uring_cqe_seen (&ring, cqe);
LOG_END (" ");
01080
01081
            return SUCCESS_CODE;
01082
01083
01084
        if (cqe->res < 0)</pre>
01085
         {
            list_delete_node (&s->list, req);
LOG_END (" ");
01086
01087
            return CRITICAL_CODE;
01088
01089
01090
        if (req->client_socket == s->efds[sel])
01091
01092
            io_uring_cqe_seen (&ring, cqe);
            list_delete_node (&s->list, req);
LOG_END (" ");
01093
01094
01095
            return ERROR_CODE;
01097
        io_uring_cqe_seen (&ring, cqe);
01098
        switch (req->event_type)
01099
01100
         case EV REA:
01101
           handle_event_read (cqe, s, req);
01102
            break;
          case EV_WRI:
01103
01104
            handle_write (s, req->client_socket);
01105
            list_delete_node (&s->list, req);
01106
            break;
01107
        LOG_END (" ");
01108
        return SUCCESS_CODE;
01109
01110 }
01111
01112 // saurion_worker_slave
01113 void
01114 saurion worker slave (void *arg)
01115 {
01116
        LOG_INIT (" ");
01117
        struct saurion_wrapper *const ss = (struct saurion_wrapper *)arg;
        struct saurion *s = ss->s;
const int sel = ss->sel;
01118
01119
01120
       free (ss);
01121
01122
        add_efd (s, s->efds[sel], sel);
01123
01124
        pthread_mutex_lock (&s->status_m);
01125
        ++s->status;
        pthread_cond_broadcast (&s->status_c);
01126
01127
        pthread_mutex_unlock (&s->status_m);
01128
        while (1)
01129
         {
01130
            int res = saurion_worker_slave_loop_it (s, sel);
             if (res == ERROR_CODE || res == CRITICAL_CODE)
01131
01132
                break;
01133
01134
01135
01136
        pthread_mutex_lock (&s->status_m);
         --s->status;
01137
        pthread_cond_signal (&s->status_c);
01138
        pthread_mutex_unlock (&s->status_m);
01139
        LOG_END (" ");
01140
01141
        return;
01142 }
01143
01144 // saurion_start
01145 [[nodiscard]]
```

7.16 low saurion.c 103

```
01146 int
01147 saurion_start (struct saurion *const s)
01148 {
01149
        threadpool_init (s->pool);
        threadpool_add (s->pool, saurion_worker_master, s);
struct saurion_wrapper *ss = NULL;
for (uint32_t i = 1; i < s->n_threads; ++i)
01150
01151
01152
01153
01154
            ss = (struct saurion_wrapper *)malloc (sizeof (struct saurion_wrapper));
            if (!ss)
{
01155
              return ERROR_CODE;
}
01156
01157
01158
            ss->s = s;
01159
01160
            ss->sel = i;
01161
            threadpool_add (s->pool, saurion_worker_slave, ss);
01162
01163
        pthread_mutex_lock (&s->status_m);
        while (s->status < (int)s->n_threads)
01164
        {
01165
01166
            pthread_cond_wait (&s->status_c, &s->status_m);
01167
01168
        pthread_mutex_unlock (&s->status_m);
01169
        return SUCCESS CODE;
01170 }
01171
01172 // saurion_stop
01173 void
01174 saurion_stop (const struct saurion *const s)
01175 {
01176
        uint64 t u = 1;
01177
        for (uint32_t i = 0; i < s->n_threads; ++i)
01178
01179
            while (write (s\rightarrow efds[i], \&u, sizeof(u)) < 0)
01180
                nanosleep (&TIMEOUT_RETRY_SPEC, NULL);
01181
01182
01183
01184
        threadpool_wait_empty (s->pool);
01185 }
01186
01187 // saurion_destroy
01188 void
01189 saurion_destroy (struct saurion *const s)
01190 {
01191
        pthread_mutex_lock (&s->status_m);
01192
        while (s->status > 0)
01193
            pthread_cond_wait (&s->status_c, &s->status_m);
01194
01195
01196
        pthread_mutex_unlock (&s->status_m);
01197
        threadpool_destroy (s->pool);
01198
        for (uint32_t i = 0; i < s->n_threads; ++i)
01199
01200
            io_uring_queue_exit (&s->rings[i]);
01201
            pthread_mutex_destroy (&s->m_rings[i]);
01202
01203
        free (s->m_rings);
01204
        list_free (&s->list);
01205
        for (uint32_t i = 0; i < s->n_threads; ++i)
01206
         {
01207
            close (s->efds[i]);
01208
01209
        free (s->efds);
01210
        if (!s->ss)
01211
         {
01212
            close (s->ss);
          }
01213
01214
        free (s->rings);
01215
        pthread_mutex_destroy (&s->status_m);
01216
        pthread_cond_destroy (&s->status_c);
01217
        free (s);
01218 }
01219
01220 // saurion_send
01221 void
01222 saurion_send (struct saurion *const s, const int fd, const char *const msg)
01223 {
01224
        add_write (s, fd, msg, next (s));
01225 }
```

7.17 / w/saurion/saurion/src/main.c File Reference

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

7.18 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 *
00007 increment (void *arg)
00008 {
00009
        int id = *((int *)arg);
00010
        for (int i = 0; i < 100000; ++i)
00011
00012
00013
             if (i % 10000 == 0)
00014
                 printf ("Thread %d at iteration %d\n", id, i);
00015
00016
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;
int id2 = 2;
00028
00030
        printf ("Starting threads...\n");
00031
00032
        if (pthread_create (&t1, NULL, increment, &id1))
00033
00034
             fprintf (stderr, "Error creating thread 1\n");
00035
             return 1;
00036
00037
        if (pthread_create (&t2, NULL, increment, &id2))
00038
         {
00039
             fprintf (stderr, "Error creating thread 2\n");
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"); return 2;
00046
00047
00048
00049
        printf ("Thread 1 joined\n");
00050
        printf ("Waiting for thread 2 to join...\n"); if (pthread_join (t2, NULL))
00051
00052
00053
00054
             fprintf (stderr, "Error joining thread 2\n");
00055
00056
00057
        printf ("Thread 2 joined\n");
00058
        printf ("Final counter value: %d\n", counter);
00059
00060
        return 0;
00061 }
```

7.19 /__w/saurion/saurion/src/saurion.cpp File Reference

```
#include "saurion.hpp"
#include "low_saurion.h"
```

7.20 saurion.cpp 105

#include <unistd.h>

Include dependency graph for saurion.cpp:

7.20 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 #include <unistd.h>
00005
00006 Saurion::Saurion (const uint32_t thds, const int sck) noexcept
00007 {
80000
        this->s = saurion_create (thds);
00009
        if (!this->s)
00010
00011
             return;
00012
00013
        this->s->ss = sck;
00014 }
00015
00016 Saurion::~Saurion ()
00017 {
00018
        close (s->ss);
00018 close (s->ss);
00019 saurion_destroy (this->s);
00020 }
00021
00022 void
00023 Saurion::init () noexcept
00024 {
        if (!saurion_start (this->s))
00026
         {
00027
             return;
00028
00029 }
00030
00031 void
00032 Saurion::stop () const noexcept
00033 {
00034
        saurion_stop (this->s);
00035 }
00036
00037 Saurion *
00038 Saurion::on_connected (Saurion::ConnectedCb ncb, void *arg) noexcept
00039 {
00040 s->cb.on_connected = ncb;
00041 s->cb.on_connected_arg = arg;
00042
        return this;
00043 }
00044
00045 Saurion *
00046 Saurion::on_readed (Saurion::ReadedCb ncb, void *arg) noexcept
00047 {
00048
        s->cb.on_readed = ncb;
00049 s->cb.on_readed_arg = arg;
00050 return this;
00051 }
00052
00053 Saurion *
00054 Saurion::on wrote (Saurion::WroteCb ncb, void *arg) noexcept
00055 {
00056 s->cb.on_wrote = ncb;
00057 s->cb.on_wrote_arg = arg;
00058 return this;
00059 }
00060
00061 Saurion *
00062 Saurion::on_closed (Saurion::ClosedCb ncb, void *arg) noexcept
00063 {
00064 s->cb.on_closed = ncb;
00065 s->cb.on_closed_arg = arg;
00066 return this;
00067 }
00068
00069 Saurion *
00070 Saurion::on_error (Saurion::ErrorCb ncb, void *arg) noexcept
00071 {
00072
        s->cb.on_error = ncb;
00073
        s->cb.on_error_arg = arg;
00074
        return this;
00075 }
```

```
00076
00077 void
00078 Saurion::send (const int fd, const char *const msg) noexcept
00079 {
00080    saurion_send (this->s, fd, msg);
00081 }
```

7.21 /_w/saurion/saurion/src/threadpool.c File Reference

```
#include "threadpool.h"
#include "config.h"
#include <nanologger.h>
#include <pthread.h>
#include <stdlib.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)

7.21.1 Macro Definition Documentation

7.21.1.1 FALSE

```
#define FALSE 0
```

Definition at line 8 of file threadpool.c.

7.22 threadpool.c 107

7.21.1.2 TRUE

```
#define TRUE 1
```

Definition at line 7 of file threadpool.c.

7.21.2 Function Documentation

7.21.2.1 threadpool_worker()

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

```
00102 {
        LOG_INIT (" ");
00103
        struct threadpool *pool = (struct threadpool *)arg;
00105
        while (TRUE)
00106
            pthread_mutex_lock (&pool->queue_lock);
while (pool->task_queue_head == NULL && !pool->stop)
00107
00108
00109
              {
00110
                pthread_cond_wait (&pool->queue_cond, &pool->queue_lock);
00111
00112
00113
            if (pool->stop && pool->task_queue_head == NULL)
00114
             {
00115
                pthread_mutex_unlock (&pool->queue_lock);
00116
                break;
00117
00118
00119
            struct task *task = pool->task_queue_head;
00120
            if (task != NULL)
00121
              {
                pool->task_queue_head = task->next;
00122
00123
                 if (pool->task_queue_head == NULL)
00124
                  pool->task_queue_tail = NULL;
00125
00126
                 if (pool->task_queue_head == NULL)
00127
00128
                     pthread_cond_signal (&pool->empty_cond);
00130
00131
            pthread_mutex_unlock (&pool->queue_lock);
00132
00133
            if (task != NULL)
00134
00135
                task->function (task->argument);
00136
                free (task);
00137
00138
       LOG_END (" ");
pthread_exit (NULL);
00139
00140
00141
        return NULL;
00142 }
```

7.22 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 <stdlib.h> // for free, malloc
```

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

7.22 threadpool.c 109

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

```
LOG_END (" ");
00181
00182
            return;
00183
00184
00185
        new_task->function = function;
        new_task->argument = argument;
00186
        new_task->next = NULL;
00187
00188
00189
        pthread_mutex_lock (&pool->queue_lock);
00190
00191
        if (pool->task_queue_head == NULL)
00192
00193
            pool->task_queue_head = new_task;
00194
            pool->task_queue_tail = new_task;
00195
00196
        else
00197
            pool->task_queue_tail->next = new_task;
00198
            pool->task_queue_tail = new_task;
00199
00200
00201
        pthread_cond_signal (&pool->queue_cond);
00202
        pthread_mutex_unlock (&pool->queue_lock);
LOG_END (" ");
00203
00204
00205 }
00206
00207 void
00208 threadpool_stop (struct threadpool *pool)
00209 {
00210
        LOG INIT (" ");
        if (pool == NULL || !pool->started)
00211
00212
         {
00213
            LOG_END (" ");
00214
            return;
00215
00216
        threadpool_wait_empty (pool);
00217
00218
        pthread_mutex_lock (&pool->queue_lock);
00219
        pool->stop = TRUE;
00220
        pthread_cond_broadcast (&pool->queue_cond);
00221
        pthread_mutex_unlock (&pool->queue_lock);
00222
        for (size_t i = 0; i < pool->num_threads; i++)
00223
00224
00225
            pthread_join (pool->threads[i], NULL);
00226
00227
        pool->started = FALSE;
00228
        LOG_END (" ");
00229 }
00230
00231 int
00232 threadpool_empty (struct threadpool *pool)
00233 {
00234
        LOG_INIT (" ");
00235
        if (pool == NULL)
00236
         {
00237
            LOG_END (" ");
00238
            return TRUE;
00239
        pthread_mutex_lock (&pool->queue_lock);
int empty = (pool->task_queue_head == NULL);
00240
00241
00242
        pthread_mutex_unlock (&pool->queue_lock);
00243
        LOG_END (" ");
00244
        return empty;
00245 }
00246
00247 void
00248 threadpool_wait_empty (struct threadpool *pool)
00249 {
00250
        LOG_INIT (" ");
00251
        if (pool == NULL)
00252
            LOG_END (" ");
00253
00254
            return:
00255
00256
        pthread_mutex_lock (&pool->queue_lock);
00257
        while (pool->task_queue_head != NULL)
00258
00259
            pthread_cond_wait (&pool->empty_cond, &pool->queue_lock);
00260
00261
        pthread_mutex_unlock (&pool->queue_lock);
        LOG_END (" ");
00262
00263 }
00264
00265 void
00266 threadpool_destroy (struct threadpool *pool)
00267 {
```

7.22 threadpool.c 111

Index

/_w/saurion/saurion/include/client_interface.hpp, 56	55,	dest_ptr, 30 len, 30
/w/saurion/saurion/include/linked_list.h, 56, 58		max_iov_cont, 31
/w/saurion/saurion/include/low_saurion.h, 58, 65		req, 31
	66 c	elean
		ClientInterface, 32
/_w/saurion/saurion/include/threadpool.h, 67, 68	С	lient_interface.hpp
/_w/saurion/saurion/src/linked_list.c, 68, 72		set_fifoname, 55
/_w/saurion/saurion/src/low_saurion.c, 73, 89		set_port, 55
/ w/saurion/saurion/src/main.c, 104	C	lient_socket
/ w/saurion/saurion/src/saurion.cpp, 104, 105		request, 36
/_w/saurion/saurion/src/threadpool.c, 106, 107	(ClientInterface, 31
_POSIX_C_SOURCE		\sim ClientInterface, 32
LowSaurion, 12		clean, 32
~ClientInterface		ClientInterface, 32
ClientInterface, 32		connect, 33
~Saurion		disconnect, 33
Saurion, 43		fifo, 34
		fifoname, 34
add_accept		getFifoPath, 33
low_saurion.c, 77		getPort, 33
add_efd		operator=, 33
low_saurion.c, 77		pid, 34
add fd		port, 34
low_saurion.c, 77		reads, 33
add read		send, 34
low_saurion.c, 78		ClosedCb
add_read_continue		Saurion, 42
low_saurion.c, 78	C	connect
add write		ClientInterface, 33
low_saurion.c, 79	(ConnectedCb
allocate_iovec		Saurion, 42
LowSaurion, 13	C	cont rem
argument		chunk_params, 29
task, 51	C	cont sz
		chunk_params, 29
calculate_max_iov_content	C	copy_data
low_saurion.c, 80		low saurion.c, 80
cb	C	reate node
low_saurion.h, 60		linked list.c, 69
saurion, 38	C	curr iov
children		chunk_params, 30
Node, 35	C	curr iov off
chunk_params, 29		chunk_params, 30
cont_rem, 29		onam_paramo, oo
cont_sz, 29	d	lest
curr_iov, 30		chunk params, 30
curr_iov_off, 30	d	lest off
dest, 30		chunk_params, 30
dest_off, 30	d	lest_ptr

chunk_params, 30	low_saurion.c, 84
disconnect	htonll
ClientInterface, 33	low_saurion.c, 84
efds	init
low_saurion.h, 60	Saurion, 44
saurion, 39	initialize_iovec
empty_cond	LowSaurion, 13
threadpool, 52	iov
ErrorCb	request, 36
Saurion, 42	iovec count
EV ACC	request, 37
low_saurion.c, 75	•
EV ERR	len
low_saurion.c, 75	chunk_params, 30
EV REA	linked_list.c
low saurion.c, 76	create_node, 69
EV WAI	free node, 69
low saurion.c, 76	list_delete_node, 70
EV WRI	list free, 70
low saurion.c, 76	list insert, 71
event type	list mutex, 71
= 71	linked list.h
request, 36	list delete node, 56
FALSE	list_free, 57
threadpool.c, 106 fifo	list_insert, 57
ClientInterface, 34	low_saurion.h, 60
fifoname	saurion, 39
ClientInterface, 34	list_delete_node
free_node	linked_list.c, 70
linked_list.c, 69	linked_list.h, 56
free_request	list_free
LowSaurion, 13	linked_list.c, 70
function	linked_list.h, 57
task, 51	list_insert
	linked_list.c, 71
getFifoPath	linked_list.h, 57
ClientInterface, 33	list_mutex
getPort	linked_list.c, 71
ClientInterface, 33	low_saurion.c
	add_accept, 77
handle_accept	add_efd, 77
low_saurion.c, 80	add_fd, 77
handle_close	add_read, 78
low_saurion.c, 81	add_read_continue, 78
handle_error	add write, 79
low_saurion.c, 81	calculate_max_iov_content, 80
handle_event_read	copy_data, 80
low_saurion.c, 81	EV ACC, 75
handle_new_message	EV ERR, 75
low_saurion.c, 82	EV REA, 76
handle_partial_message	EV WAI, 76
low_saurion.c, 82	EV_VVAI, 76 EV WRI, 76
handle_previous_message	_
low_saurion.c, 83	handle_accept, 80
handle_read	handle_close, 81
low_saurion.c, 83	handle_error, 81
handle_write	handle_event_read, 81
	handle_new_message, 82

	handle_partial_message, 82	low_saurion.c, 76
	handle_previous_message, 83	max_iov_cont
	handle_read, 83	chunk_params, 31
	handle_write, 84	MIN
	htonll, 84	low_saurion.c, 76
	MAX, 76	
	MIN, 76	n_threads
	next, 85	low_saurion.h, 60
	ntohll, 85	saurion, 39
	prepare_destination, 85	next
	read_chunk_free, 85	low_saurion.c, 85 low_saurion.h, 60
	saurion_worker_master, 86	Node, 35
	saurion_worker_master_loop_it, 86	saurion, 39
	saurion_worker_slave, 87	task, 51
	saurion_worker_slave_loop_it, 88	next iov
	TIMEOUT_RETRY_SPEC, 89	request, 37
love	validate_and_update, 88	next offset
IOW_	saurion.h	request, 37
	cb, 60	Node, 35
	efds, 60 list, 60	children, 35
	m_rings, 60	next, 35
	n_threads, 60	ptr, 35
	next, 60	size, 35
	on_closed, 61	ntohll
	on_closed_arg, 61	low saurion.c, 85
	on_connected, 61	num_threads
	on_connected_arg, 62	threadpool, 52
	on_error, 62	,
	on_error_arg, 62	on_closed
	on_readed, 62	low_saurion.h, 61
	on_readed_arg, 63	Saurion, 44
	on_wrote, 63	saurion_callbacks, 47
	on_wrote_arg, 63	on_closed_arg
	pool, 63	low_saurion.h, 61
	rings, 64	saurion_callbacks, 47
	ss, 64	on_connected
	status, 64	low_saurion.h, 61
	status_c, 64	Saurion, 44
	status m, 64	saurion_callbacks, 47
Low	Saurion, 9	on_connected_arg
	_POSIX_C_SOURCE, 12	low_saurion.h, 62
	allocate_iovec, 13	saurion_callbacks, 48
	free_request, 13	on_error
	initialize_iovec, 13	low_saurion.h, 62
	PACKING_SZ, 12	Saurion, 44
	read_chunk, 15	saurion_callbacks, 48
	saurion_create, 16	on_error_arg
	saurion_destroy, 18	low_saurion.h, 62
	saurion_send, 19	saurion_callbacks, 48
	saurion_set_socket, 19	on_readed
	saurion_start, 20	low_saurion.h, 62
	saurion_stop, 21	Saurion, 44
	set_request, 21	saurion_callbacks, 48
		on_readed_arg
m_ri	_	low_saurion.h, 63
	low_saurion.h, 60	saurion_callbacks, 49
1111	saurion, 39	on_wrote
MAX	•	low_saurion.h, 63

Saurion, 45	Saurion, 46
saurion_callbacks, 49	saurion_wrapper, 50
on_wrote_arg	Saurion, 41
low_saurion.h, 63	\sim Saurion, 43
saurion callbacks, 49	ClosedCb, 42
operator=	ConnectedCb, 42
ClientInterface, 33	ErrorCb, 42
Saurion, 45	init, 44
	on_closed, 44
PACKING SZ	on connected, 44
LowSaurion, 12	on error, 44
pid	_ ,
ClientInterface, 34	on_readed, 44
	on_wrote, 45
pool	operator=, 45
low_saurion.h, 63	ReadedCb, 42
saurion, 40	s, 46
port	Saurion, 43
ClientInterface, 34	send, 45
prepare_destination	stop, 45
low_saurion.c, 85	WroteCb, 42
prev	saurion, 38
request, 37	cb, 38
prev_remain	efds, 39
request, 37	list, 39
prev_size	m_rings, 39
request, 37	
·	n_threads, 39
ptr Node 35	next, 39
Node, 35	pool, 40
quous sand	rings, 40
queue_cond	ss, 40
threadpool, 52	status, 40
queue_lock	status_c, 40
threadpool, 52	status_m, 41
	saurion callbacks, 46
read_chunk	on_closed, 47
LowSaurion, 15	on closed arg, 47
read_chunk_free	on connected, 47
low_saurion.c, 85	-
ReadedCb	on_connected_arg, 48
Saurion, 42	on_error, 48
reads	on_error_arg, 48
ClientInterface, 33	on_readed, 48
req	on_readed_arg, 49
chunk params, 31	on_wrote, 49
request, 36	on_wrote_arg, 49
•	saurion_create
client_socket, 36	LowSaurion, 16
event_type, 36	saurion destroy
iov, 36	LowSaurion, 18
iovec_count, 37	saurion_send
next_iov, 37	LowSaurion, 19
next_offset, 37	saurion_set_socket
prev, 37	
prev_remain, 37	LowSaurion, 19
prev size, 37	saurion_start
rings	LowSaurion, 20
low_saurion.h, 64	saurion_stop
saurion, 40	LowSaurion, 21
Jaaron, To	
	saurion_worker_master
S	saurion_worker_master low_saurion.c, 86

saurion_worker_master_loop_it	num_threads, 52
low_saurion.c, 86	queue_cond, 52
saurion_worker_slave	queue_lock, 52
low_saurion.c, 87	started, 52
saurion_worker_slave_loop_it	stop, 52
low_saurion.c, 88	task_queue_head, 53
saurion_wrapper, 50	task_queue_tail, 53
s, 50	threads, 53
sel, 50	threadpool.c
sel	FALSE, 106
saurion_wrapper, 50	threadpool_worker, 107
send	TRUE, 106
ClientInterface, 34	threadpool_add
Saurion, 45	ThreadPool, 23
set_fifoname	threadpool_create
client_interface.hpp, 55	ThreadPool, 24
set_port	threadpool_create_default
client_interface.hpp, 55	ThreadPool, 25
set_request	threadpool_destroy
LowSaurion, 21	ThreadPool, 25
size	threadpool_empty
Node, 35	ThreadPool, 25
SS	threadpool_init
low_saurion.h, 64	ThreadPool, 26
saurion, 40	threadpool_stop
started	ThreadPool, 26
threadpool, 52	threadpool_wait_empty
status	ThreadPool, 27
low_saurion.h, 64	threadpool_worker
saurion, 40	threadpool.c, 107
status_c	threads
low_saurion.h, 64	threadpool, 53
saurion, 40	TIMEOUT_RETRY_SPEC
status_m	low_saurion.c, 89
low_saurion.h, 64	TRUE
saurion, 41	threadpool.c, 106
stop	validate and undate
Saurion, 45	validate_and_update
threadpool, 52	low_saurion.c, 88
task, 50	WroteCb
argument, 51	Saurion, 42
function, 51	Gadrion, 42
next, 51	
task queue head	
threadpool, 53	
task queue tail	
threadpool, 53	
ThreadPool, 23	
threadpool_add, 23	
threadpool_create, 24	
threadpool_create_default, 25	
threadpool_destroy, 25	
threadpool_destroy, 25 threadpool_empty, 25	
threadpool_init, 26	
threadpool_stop, 26	
threadpool_wait_empty, 27	
threadpool, 51	
empty_cond, 52	