

# LWT Threads

1.0

Generated by Doxygen 1.8.8

Mon May 4 2015 13:33:59



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	http_req Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.1.2	Field Documentation . . . . .	5
3.1.2.1	fd . . . . .	5
3.1.2.2	path . . . . .	5
3.1.2.3	req_len . . . . .	5
3.1.2.4	request . . . . .	5
3.1.2.5	resp_hd_len . . . . .	5
3.1.2.6	resp_head . . . . .	5
3.1.2.7	resp_len . . . . .	6
3.1.2.8	response . . . . .	6
3.2	kthd_event Struct Reference . . . . .	6
3.2.1	Field Documentation . . . . .	6
3.2.1.1	block . . . . .	6
3.2.1.2	channel . . . . .	6
3.2.1.3	group . . . . .	6
3.2.1.4	is_done . . . . .	6
3.2.1.5	kthd . . . . .	6
3.2.1.6	lwt . . . . .	6
3.2.1.7	op . . . . .	6
3.2.1.8	originator . . . . .	6
3.3	lwt_cgrp_t Struct Reference . . . . .	7

3.3.1	Detailed Description	7
3.3.2	Member Function Documentation	7
3.3.2.1	LIST_HEAD	7
3.3.2.2	TAILQ_HEAD	7
3.3.3	Field Documentation	7
3.3.3.1	creator_thread	7
3.3.3.2	waiting_thread	7
3.4	lwt_chan_t Struct Reference	8
3.4.1	Detailed Description	8
3.4.2	Member Function Documentation	8
3.4.2.1	LIST_ENTRY	8
3.4.2.2	LIST_ENTRY	8
3.4.2.3	LIST_HEAD	8
3.4.2.4	TAILQ_ENTRY	9
3.4.2.5	TAILQ_HEAD	9
3.4.3	Field Documentation	9
3.4.3.1	async_buffer	9
3.4.3.2	buffer_size	9
3.4.3.3	channel_group	9
3.4.3.4	end_index	9
3.4.3.5	kthd	9
3.4.3.6	mark	9
3.4.3.7	num_entries	9
3.4.3.8	receiver	9
3.4.3.9	snd_cnt	9
3.4.3.10	start_index	10
3.4.3.11	sync_buffer	10
3.5	lwt_kthd_data Struct Reference	10
3.5.1	Field Documentation	10
3.5.1.1	channel	10
3.5.1.2	channel_fn	10
3.5.1.3	flags	10
3.5.1.4	parent	10
3.5.1.5	ready	10
3.6	lwt_kthd_t Struct Reference	10
3.6.1	Member Function Documentation	11
3.6.1.1	LIST_HEAD	11

3.6.1.2	TAILQ_HEAD	11
3.6.2	Field Documentation	11
3.6.2.1	blocked_cv	11
3.6.2.2	blocked_mutex	11
3.6.2.3	buffer_head	11
3.6.2.4	buffer_tail	11
3.6.2.5	buffer_thread	11
3.6.2.6	event_buffer	11
3.6.2.7	is_blocked	12
3.6.2.8	pthread	12
3.7	lwt_t Struct Reference	12
3.7.1	Detailed Description	12
3.7.2	Member Function Documentation	13
3.7.2.1	LIST_ENTRY	13
3.7.2.2	LIST_ENTRY	13
3.7.2.3	LIST_ENTRY	13
3.7.2.4	LIST_ENTRY	13
3.7.2.5	LIST_ENTRY	13
3.7.2.6	LIST_HEAD	13
3.7.2.7	LIST_HEAD	13
3.7.2.8	TAILQ_ENTRY	13
3.7.2.9	TAILQ_ENTRY	13
3.7.3	Field Documentation	13
3.7.3.1	args	13
3.7.3.2	flags	14
3.7.3.3	id	14
3.7.3.4	info	14
3.7.3.5	kthd	14
3.7.3.6	max_addr_thread_stack	14
3.7.3.7	min_addr_thread_stack	14
3.7.3.8	parent	14
3.7.3.9	return_value	14
3.7.3.10	start_routine	14
3.7.3.11	sync_buffer	14
3.7.3.12	thread_sp	14
3.8	msort_args Struct Reference	15
3.8.1	Detailed Description	15

3.8.2	Field Documentation	15
3.8.2.1	begin_index	15
3.8.2.2	data	15
3.8.2.3	end_index	15
3.8.2.4	swap	15
<b>4</b>	<b>File Documentation</b>	<b>17</b>
4.1	cas.h File Reference	17
4.2	content.c File Reference	17
4.2.1	Macro Definition Documentation	17
4.2.1.1	MAX_CONTENT_SZ	17
4.2.2	Function Documentation	17
4.2.2.1	content_get	18
4.2.2.2	error_resp	18
4.2.2.3	sanity_check	18
4.3	content.h File Reference	18
4.3.1	Function Documentation	18
4.3.1.1	content_get	18
4.4	enums.h File Reference	18
4.4.1	Enumeration Type Documentation	18
4.4.1.1	lwt_flags_t	18
4.4.1.2	lwt_info_t	19
4.4.1.3	lwt_remote_op_t	19
4.5	faa.c File Reference	19
4.5.1	Function Documentation	20
4.5.1.1	fetch_and_add	20
4.6	faa.h File Reference	20
4.6.1	Function Documentation	20
4.6.1.1	fetch_and_add	20
4.7	kthd_server.c File Reference	21
4.7.1	Macro Definition Documentation	22
4.7.1.1	LWT_CACHE	22
4.7.1.2	MAX_ACCEPTORS	22
4.7.1.3	MAX_CACHE_ENTRIES	22
4.7.1.4	MAX_REQ_SZ	22
4.7.1.5	POOL_SIZE	22
4.7.2	Function Documentation	22

4.7.2.1	<a href="#">accept_worker</a>	22
4.7.2.2	<a href="#">newfd_create_req_kthd</a>	22
4.7.2.3	<a href="#">process_kthd_server</a>	22
4.7.2.4	<a href="#">read_cache</a>	22
4.7.2.5	<a href="#">read_cache_kthd</a>	23
4.7.2.6	<a href="#">read_fs</a>	23
4.7.2.7	<a href="#">respond_and_free_req_kthd</a>	23
4.7.2.8	<a href="#">spawn_fs_workers</a>	23
4.8	<a href="#">kthd_server.h File Reference</a>	23
4.8.1	<a href="#">Function Documentation</a>	24
4.8.1.1	<a href="#">process_kthd_server</a>	24
4.9	<a href="#">lwt.c File Reference</a>	25
4.9.1	<a href="#">Macro Definition Documentation</a>	26
4.9.1.1	<a href="#">DEFAULT_ID</a>	26
4.9.1.2	<a href="#">INIT_ID</a>	26
4.9.1.3	<a href="#">POOL_SIZE</a>	26
4.9.2	<a href="#">Function Documentation</a>	27
4.9.2.1	<a href="#">__attribute__</a>	27
4.9.2.2	<a href="#">__attribute__</a>	27
4.9.2.3	<a href="#">__cleanup_joined_thread</a>	27
4.9.2.4	<a href="#">__init_lwt_main</a>	27
4.9.2.5	<a href="#">__init_new_lwt</a>	27
4.9.2.6	<a href="#">__insert_runnable_tail</a>	27
4.9.2.7	<a href="#">__lwt_dispatch</a>	27
4.9.2.8	<a href="#">__lwt_schedule</a>	28
4.9.2.9	<a href="#">__lwt_stack_get</a>	28
4.9.2.10	<a href="#">__lwt_stack_return</a>	28
4.9.2.11	<a href="#">__lwt_trampoline</a>	28
4.9.2.12	<a href="#">__reinit_lwt</a>	28
4.9.2.13	<a href="#">LIST_HEAD</a>	28
4.9.2.14	<a href="#">lwt_block</a>	28
4.9.2.15	<a href="#">lwt_create</a>	29
4.9.2.16	<a href="#">lwt_current</a>	29
4.9.2.17	<a href="#">lwt_die</a>	29
4.9.2.18	<a href="#">lwt_id</a>	29
4.9.2.19	<a href="#">lwt_info</a>	29
4.9.2.20	<a href="#">lwt_join</a>	30

4.9.2.21	<code>lwt_signal</code>	30
4.9.2.22	<code>lwt_yield</code>	30
4.9.3	Variable Documentation	30
4.9.3.1	<code>current_thread</code>	30
4.9.3.2	<code>original_thread</code>	30
4.10	<code>lwt.h</code> File Reference	30
4.10.1	Function Documentation	31
4.10.1.1	<code>__destroy__</code>	31
4.10.1.2	<code>__init__</code>	31
4.10.1.3	<code>lwt_block</code>	31
4.10.1.4	<code>lwt_create</code>	31
4.10.1.5	<code>lwt_current</code>	32
4.10.1.6	<code>lwt_die</code>	32
4.10.1.7	<code>lwt_id</code>	32
4.10.1.8	<code>lwt_info</code>	32
4.10.1.9	<code>lwt_join</code>	32
4.10.1.10	<code>lwt_signal</code>	32
4.10.1.11	<code>lwt_yield</code>	33
4.11	<code>lwt_cgrp.c</code> File Reference	33
4.11.1	Function Documentation	34
4.11.1.1	<code>__init_event</code>	34
4.11.1.2	<code>__remove_event</code>	34
4.11.1.3	<code>lwt_cgrp</code>	34
4.11.1.4	<code>lwt_cgrp_add</code>	34
4.11.1.5	<code>lwt_cgrp_free</code>	34
4.11.1.6	<code>lwt_cgrp_rem</code>	35
4.11.1.7	<code>lwt_cgrp_wait</code>	35
4.11.1.8	<code>lwt_chan_mark_get</code>	35
4.11.1.9	<code>lwt_chan_mark_set</code>	35
4.12	<code>lwt_cgrp.h</code> File Reference	36
4.12.1	Function Documentation	36
4.12.1.1	<code>__init_event</code>	36
4.12.1.2	<code>__remove_event</code>	36
4.12.1.3	<code>lwt_cgrp</code>	37
4.12.1.4	<code>lwt_cgrp_add</code>	37
4.12.1.5	<code>lwt_cgrp_free</code>	37
4.12.1.6	<code>lwt_cgrp_rem</code>	37



4.12.1.7	<a href="#">lwt_cgrp_wait</a>	38
4.12.1.8	<a href="#">lwt_chan_mark_get</a>	38
4.12.1.9	<a href="#">lwt_chan_mark_set</a>	38
4.13	<a href="#">lwt_chan.c File Reference</a>	38
4.13.1	<a href="#">Function Documentation</a>	39
4.13.1.1	<a href="#">__insert_blocked_sender_to_chan</a>	39
4.13.1.2	<a href="#">__insert_sender_to_chan</a>	39
4.13.1.3	<a href="#">__pop_data_from_async_buffer</a>	40
4.13.1.4	<a href="#">__remove_blocked_sender_from_chan</a>	40
4.13.1.5	<a href="#">__remove_sender_from_chan</a>	40
4.13.1.6	<a href="#">lwt_chan</a>	40
4.13.1.7	<a href="#">lwt_chan_deref</a>	40
4.13.1.8	<a href="#">lwt_create_chan</a>	41
4.13.1.9	<a href="#">lwt_rcv</a>	41
4.13.1.10	<a href="#">lwt_rcv_chan</a>	41
4.13.1.11	<a href="#">lwt_snd</a>	41
4.13.1.12	<a href="#">lwt_snd_chan</a>	42
4.14	<a href="#">lwt_chan.h File Reference</a>	42
4.14.1	<a href="#">Function Documentation</a>	43
4.14.1.1	<a href="#">__insert_blocked_sender_to_chan</a>	43
4.14.1.2	<a href="#">__insert_sender_to_chan</a>	44
4.14.1.3	<a href="#">__remove_blocked_sender_from_chan</a>	44
4.14.1.4	<a href="#">__remove_sender_from_chan</a>	44
4.14.1.5	<a href="#">lwt_chan</a>	44
4.14.1.6	<a href="#">lwt_chan_deref</a>	44
4.14.1.7	<a href="#">lwt_create_chan</a>	45
4.14.1.8	<a href="#">lwt_rcv</a>	45
4.14.1.9	<a href="#">lwt_rcv_chan</a>	45
4.14.1.10	<a href="#">lwt_snd</a>	45
4.14.1.11	<a href="#">lwt_snd_chan</a>	46
4.15	<a href="#">lwt_kthd.c File Reference</a>	46
4.15.1	<a href="#">Function Documentation</a>	47
4.15.1.1	<a href="#">__get_kthd</a>	47
4.15.1.2	<a href="#">__init_kthd</a>	47
4.15.1.3	<a href="#">__init_kthd_event</a>	47
4.15.1.4	<a href="#">__lwt_buffer</a>	47
4.15.1.5	<a href="#">__pop_from_buffer</a>	47

4.15.1.6	<a href="#">__push_to_buffer</a>	48
4.15.1.7	<a href="#">lwt_kthd_create</a>	48
4.15.1.8	<a href="#">pthread_function</a>	48
4.15.2	Variable Documentation	48
4.15.2.1	<a href="#">pthread_kthd</a>	48
4.16	<a href="#">lwt_kthd.h</a> File Reference	49
4.16.1	Function Documentation	49
4.16.1.1	<a href="#">__get_kthd</a>	49
4.16.1.2	<a href="#">__init_kthd</a>	49
4.16.1.3	<a href="#">__init_kthd_event</a>	49
4.16.1.4	<a href="#">__insert_lwt_into_tail</a>	50
4.16.1.5	<a href="#">__lwt_buffer</a>	50
4.16.1.6	<a href="#">__remove_lwt_from_kthd</a>	50
4.16.1.7	<a href="#">lwt_kthd_create</a>	50
4.17	<a href="#">main.c</a> File Reference	50
4.17.1	Macro Definition Documentation	51
4.17.1.1	<a href="#">BUFFER_LENGTH</a>	51
4.17.1.2	<a href="#">MAX_CONCURRENCY</a>	51
4.17.1.3	<a href="#">MAX_DATA_SZ</a>	51
4.17.2	Enumeration Type Documentation	51
4.17.2.1	<a href="#">server_type_t</a>	51
4.17.3	Function Documentation	51
4.17.3.1	<a href="#">main</a>	51
4.17.3.2	<a href="#">server_single_request</a>	51
4.18	<a href="#">main3.c</a> File Reference	51
4.18.1	Macro Definition Documentation	52
4.18.1.1	<a href="#">GRPSZ</a>	52
4.18.1.2	<a href="#">IS_RESET</a>	52
4.18.1.3	<a href="#">ITER</a>	53
4.18.1.4	<a href="#">rdtscll</a>	53
4.18.2	Function Documentation	53
4.18.2.1	<a href="#">fn_async_steam</a>	53
4.18.2.2	<a href="#">fn_bounce</a>	53
4.18.2.3	<a href="#">fn_chan</a>	53
4.18.2.4	<a href="#">fn_grpwait</a>	53
4.18.2.5	<a href="#">fn_identity</a>	53
4.18.2.6	<a href="#">fn_join</a>	53

4.18.2.7	<a href="#">fn_nested_joins</a>	53
4.18.2.8	<a href="#">fn_null</a>	53
4.18.2.9	<a href="#">fn_sequence</a>	53
4.18.2.10	<a href="#">fn_sndr</a>	53
4.18.2.11	<a href="#">fn_sndr_1</a>	53
4.18.2.12	<a href="#">fn_sndr_2</a>	53
4.18.2.13	<a href="#">main</a>	53
4.18.2.14	<a href="#">test_crt_join_sched</a>	53
4.18.2.15	<a href="#">test_grpwait</a>	53
4.18.2.16	<a href="#">test_multisend</a>	53
4.18.2.17	<a href="#">test_perf</a>	53
4.18.2.18	<a href="#">test_perf_async_steam</a>	53
4.18.2.19	<a href="#">test_perf_channels</a>	53
4.18.3	<a href="#">Variable Documentation</a>	53
4.18.3.1	<a href="#">curr</a>	53
4.18.3.2	<a href="#">sched</a>	54
4.19	<a href="#">main_chan.c File Reference</a>	54
4.19.1	<a href="#">Macro Definition Documentation</a>	54
4.19.1.1	<a href="#">ITER</a>	54
4.19.1.2	<a href="#">MERGE_SZ</a>	54
4.19.2	<a href="#">Function Documentation</a>	55
4.19.2.1	<a href="#">child_multiple_channels</a>	55
4.19.2.2	<a href="#">child_ping</a>	55
4.19.2.3	<a href="#">child_pong</a>	55
4.19.2.4	<a href="#">main</a>	55
4.19.2.5	<a href="#">merge_sort_test</a>	55
4.19.2.6	<a href="#">msort</a>	55
4.19.2.7	<a href="#">multiple_channels_test</a>	56
4.19.2.8	<a href="#">multiple_channels_test_v2</a>	56
4.19.2.9	<a href="#">multiple_channels_test_v3</a>	56
4.19.2.10	<a href="#">ping_pong_test</a>	56
4.20	<a href="#">main_kthd.c File Reference</a>	56
4.20.1	<a href="#">Macro Definition Documentation</a>	56
4.20.1.1	<a href="#">GRPSZ</a>	56
4.20.1.2	<a href="#">ITER</a>	56
4.20.1.3	<a href="#">MAX_PING_PONG_VALUE</a>	56
4.20.2	<a href="#">Function Documentation</a>	56

4.20.2.1	fn_grpwait	56
4.20.2.2	kthd_ping	56
4.20.2.3	kthd_ping_pong_sync	56
4.20.2.4	main	57
4.20.2.5	test_grpwait	57
4.21	main_orig.c File Reference	57
4.21.1	Macro Definition Documentation	57
4.21.1.1	IS_RESET	57
4.21.1.2	ITER	58
4.21.1.3	rdtscll	58
4.21.2	Function Documentation	58
4.21.2.1	fn_bounce	58
4.21.2.2	fn_identity	58
4.21.2.3	fn_join	58
4.21.2.4	fn_nested_joins	58
4.21.2.5	fn_null	58
4.21.2.6	fn_sequence	58
4.21.2.7	main	58
4.21.2.8	test_crt_join_sched	58
4.21.2.9	test_perf	58
4.21.3	Variable Documentation	58
4.21.3.1	curr	58
4.21.3.2	sched	58
4.22	objects.h File Reference	58
4.22.1	Macro Definition Documentation	59
4.22.1.1	DEBUG	59
4.22.1.2	EVENT_BUFFER_SIZE	59
4.22.1.3	LWT_NULL	59
4.22.1.4	NUM_PAGES	59
4.22.1.5	PAGE_SIZE	59
4.22.1.6	STACK_SIZE	59
4.22.2	Typedef Documentation	59
4.22.2.1	lwt_chan_fn_t	59
4.22.2.2	lwt_fnt_t	59
4.23	server.c File Reference	60
4.23.1	Function Documentation	60
4.23.1.1	server_accept	60

4.23.1.2	server_create	60
4.24	server.h File Reference	60
4.24.1	Function Documentation	60
4.24.1.1	server_accept	60
4.24.1.2	server_create	60
4.25	simple_http.c File Reference	60
4.25.1	Macro Definition Documentation	61
4.25.1.1	MAX_DIGITS	61
4.25.2	Function Documentation	61
4.25.2.1	shttp_alloc_req	61
4.25.2.2	shttp_alloc_response_head	61
4.25.2.3	shttp_free_req	61
4.25.2.4	shttp_get_path	61
4.26	simple_http.h File Reference	61
4.26.1	Function Documentation	62
4.26.1.1	shttp_alloc_req	62
4.26.1.2	shttp_alloc_response_head	62
4.26.1.3	shttp_free_req	62
4.26.1.4	shttp_get_path	62
4.27	util.c File Reference	62
4.27.1	Macro Definition Documentation	62
4.27.1.1	MAX_REQ_SZ	62
4.27.2	Function Documentation	62
4.27.2.1	client_process	62
4.27.2.2	newfd_create_req	63
4.27.2.3	respond_and_free_req	63
4.28	util.h File Reference	63
4.28.1	Function Documentation	63
4.28.1.1	client_process	63



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">http_req</a>	5
<a href="#">kthd_event</a>	6
<a href="#">lwt_cgrp_t</a>	
Channel group for handling events within a group	7
<a href="#">lwt_chan_t</a>	
The channel for synchronous and asynchronous communication	8
<a href="#">lwt_kthd_data</a>	10
<a href="#">lwt_kthd_t</a>	10
<a href="#">lwt_t</a>	
The Lightweight Thread (LWT) struct	12
<a href="#">msort_args</a>	
Struct for passing the args to merge sort around	15





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

cas.h	17
content.c	17
content.h	18
enums.h	18
faa.c	19
faa.h	20
kthd_server.c	21
kthd_server.h	23
lwt.c	25
lwt.h	30
lwt_cgrp.c	33
lwt_cgrp.h	36
lwt_chan.c	38
lwt_chan.h	42
lwt_kthd.c	46
lwt_kthd.h	49
main.c	50
main3.c	51
main_chan.c	54
main_kthd.c	56
main_orig.c	57
objects.h	58
server.c	60
server.h	60
simple_http.c	60
simple_http.h	61
util.c	62
util.h	63



## Chapter 3

# Data Structure Documentation

### 3.1 http\_req Struct Reference

```
#include <simple_http.h>
```

#### Data Fields

- int `fd`
- char \* `request`
- int `req_len`
- char \* `path`
- char \* `resp_head`
- char \* `response`
- int `resp_hd_len`
- int `resp_len`

#### 3.1.1 Detailed Description

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

#### 3.1.2 Field Documentation

3.1.2.1 int http\_req::fd

3.1.2.2 char\* http\_req::path

3.1.2.3 int http\_req::req\_len

3.1.2.4 char\* http\_req::request

3.1.2.5 int http\_req::resp\_hd\_len

3.1.2.6 char\* http\_req::resp\_head

3.1.2.7 `int http_req::resp_len`

3.1.2.8 `char * http_req::response`

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

- [simple\\_http.h](#)

## 3.2 kthd\_event Struct Reference

```
#include <objects.h>
```

### Data Fields

- `lwt_t` [originator](#)
- `lwt_t` [lwt](#)
- `lwt_chan_t` [channel](#)
- `lwt_cgrp_t` [group](#)
- `lwt_kthd_t` [kthd](#)
- `int` [is\\_done](#)
- `int` [block](#)
- `lwt_remote_op_t` [op](#)

### 3.2.1 Field Documentation

3.2.1.1 `int kthd_event::block`

3.2.1.2 `lwt_chan_t kthd_event::channel`

3.2.1.3 `lwt_cgrp_t kthd_event::group`

3.2.1.4 `int kthd_event::is_done`

3.2.1.5 `lwt_kthd_t kthd_event::kthd`

3.2.1.6 `lwt_t kthd_event::lwt`

3.2.1.7 `lwt_remote_op_t kthd_event::op`

3.2.1.8 `lwt_t kthd_event::originator`

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

- [objects.h](#)

## 3.3 lwt\_cgrp\_t Struct Reference

Channel group for handling events within a group.

```
#include <objects.h>
```

### Public Member Functions

- [LIST\\_HEAD](#) (head\_channels\_in\_group, [lwt\\_channel](#)) head\_channels\_in\_group
- [TAILQ\\_HEAD](#) (head\_event, [lwt\\_channel](#)) head\_event

### Data Fields

- [lwt\\_t](#) [waiting\\_thread](#)
- [lwt\\_t](#) [creator\\_thread](#)

#### 3.3.1 Detailed Description

Channel group for handling events within a group.

#### 3.3.2 Member Function Documentation

##### 3.3.2.1 lwt\_cgrp\_t::LIST\_HEAD ( head\_channels\_in\_group , lwt\_channel )

Definition of the head in the channels

##### 3.3.2.2 lwt\_cgrp\_t::TAILQ\_HEAD ( head\_event , lwt\_channel )

Definition of the head node for the event queue

#### 3.3.3 Field Documentation

##### 3.3.3.1 lwt\_t lwt\_cgrp\_t::creator\_thread

Creator thread

##### 3.3.3.2 lwt\_t lwt\_cgrp\_t::waiting\_thread

Waiting thread

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

- [objects.h](#)

### 3.4 lwt\_chan\_t Struct Reference

The channel for synchronous and asynchronous communication.

```
#include <objects.h>
```

#### Public Member Functions

- [LIST\\_HEAD](#) (head\_senders, [lwt](#)) head\_senders
- [TAILQ\\_HEAD](#) (head\_blocked\_senders, [lwt](#)) head\_blocked\_senders
- [LIST\\_ENTRY](#) ([lwt\\_channel](#)) receiver\_channels
- [LIST\\_ENTRY](#) ([lwt\\_channel](#)) channels\_in\_group
- [TAILQ\\_ENTRY](#) ([lwt\\_channel](#)) events

#### Data Fields

- int [snd\\_cnt](#)
- [lwt\\_t](#) [receiver](#)
- void \* [sync\\_buffer](#)
- void \*\* [async\\_buffer](#)
- unsigned volatile int [start\\_index](#)
- unsigned volatile int [end\\_index](#)
- unsigned int [num\\_entries](#)
- unsigned int [buffer\\_size](#)
- [lwt\\_cgrp\\_t](#) [channel\\_group](#)
- void \* [mark](#)
- [lwt\\_kthd\\_t](#) [kthd](#)

#### 3.4.1 Detailed Description

The channel for synchronous and asynchronous communication.

#### 3.4.2 Member Function Documentation

##### 3.4.2.1 lwt\_chan\_t::LIST\_ENTRY ( lwt\_channel )

List of receiver channels in a lwt

##### 3.4.2.2 lwt\_chan\_t::LIST\_ENTRY ( lwt\_channel )

Channels in group entries

##### 3.4.2.3 lwt\_chan\_t::LIST\_HEAD ( head\_senders , lwt )

Definition of the senders head pointer

#### 3.4.2.4 lwt\_chan\_t::TAILQ\_ENTRY ( lwt\_channel )

Channels in event

#### 3.4.2.5 lwt\_chan\_t::TAILQ\_HEAD ( head\_blocked\_senders , lwt )

Definition of the blocked senders head pointer

### 3.4.3 Field Documentation

#### 3.4.3.1 void\*\* lwt\_chan\_t::async\_buffer

Async Buffer to be passed to the channel

#### 3.4.3.2 unsigned int lwt\_chan\_t::buffer\_size

Size of the buffer

#### 3.4.3.3 lwt\_cgrp\_t lwt\_chan\_t::channel\_group

Channel group

#### 3.4.3.4 unsigned volatile int lwt\_chan\_t::end\_index

End index of the buffer

#### 3.4.3.5 lwt\_kthd\_t lwt\_chan\_t::kthd

Kthd of the receiver

#### 3.4.3.6 void\* lwt\_chan\_t::mark

Mark for channel

#### 3.4.3.7 unsigned int lwt\_chan\_t::num\_entries

Num entries

#### 3.4.3.8 lwt\_t lwt\_chan\_t::receiver

The receiving thread

#### 3.4.3.9 int lwt\_chan\_t::snd\_cnt

The number of senders

#### 3.4.3.10 unsigned volatile int lwt\_chan\_t::start\_index

Start index of the buffer

#### 3.4.3.11 void\* lwt\_chan\_t::sync\_buffer

Sync buffer to be passed to the channel

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

- [objects.h](#)

### 3.5 lwt\_kthd\_data Struct Reference

```
#include <objects.h>
```

#### Data Fields

- [lwt\\_chan\\_fn\\_t](#) channel\_fn
- [lwt\\_chan\\_t](#) channel
- [lwt\\_flags\\_t](#) flags
- [lwt\\_t](#) parent
- [int](#) ready

#### 3.5.1 Field Documentation

##### 3.5.1.1 lwt\_chan\_t lwt\_kthd\_data::channel

##### 3.5.1.2 lwt\_chan\_fn\_t lwt\_kthd\_data::channel\_fn

##### 3.5.1.3 lwt\_flags\_t lwt\_kthd\_data::flags

##### 3.5.1.4 lwt\_t lwt\_kthd\_data::parent

##### 3.5.1.5 int lwt\_kthd\_data::ready

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

- [objects.h](#)

### 3.6 lwt\_kthd\_t Struct Reference

```
#include <objects.h>
```

#### Public Member Functions

- [LIST\\_HEAD](#) (head\_lwts\_in\_kthd, [lwt](#)) head\_lwts\_in\_kthd
- [TAILQ\\_HEAD](#) (head\_runnable\_threads, [lwt](#)) head\_runnable\_threads



## Data Fields

- pthread\_t [pthread](#)
- int [is\\_blocked](#)
- pthread\_mutex\_t [blocked\\_mutex](#)
- pthread\_cond\_t [blocked\\_cv](#)
- lwt\_t [buffer\\_thread](#)
- struct [kthd\\_event](#) \* [event\\_buffer](#) [[EVENT\\_BUFFER\\_SIZE](#)]
- volatile unsigned int [buffer\\_head](#)
- volatile unsigned int [buffer\\_tail](#)

### 3.6.1 Member Function Documentation

#### 3.6.1.1 lwt\_kthd\_t::LIST\_HEAD ( head\_lwts\_in\_kthd , lwt )

Point to the head of the list of lwts associated with a kthd

#### 3.6.1.2 lwt\_kthd\_t::TAILQ\_HEAD ( head\_runnable\_threads , lwt )

Pointer to the head of the run queue

### 3.6.2 Field Documentation

#### 3.6.2.1 pthread\_cond\_t lwt\_kthd\_t::blocked\_cv

Condition variable for the lwt buffer thread

#### 3.6.2.2 pthread\_mutex\_t lwt\_kthd\_t::blocked\_mutex

Mutex for the blocked lwt buffer thread

#### 3.6.2.3 volatile unsigned int lwt\_kthd\_t::buffer\_head

Head of the buffer

#### 3.6.2.4 volatile unsigned int lwt\_kthd\_t::buffer\_tail

Tail of the buffer

#### 3.6.2.5 lwt\_t lwt\_kthd\_t::buffer\_thread

Buffer thread for the lwt

#### 3.6.2.6 struct kthd\_event\* lwt\_kthd\_t::event\_buffer[EVENT\_BUFFER\_SIZE]

Event buffer for remote communication

### 3.6.2.7 `int lwt_kthd_t::is_blocked`

Status flag for if the current remote thread is blocked

### 3.6.2.8 `pthread_t lwt_kthd_t::pthread`

The Pthread belonging to the kthd

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

- [objects.h](#)

## 3.7 `lwt_t` Struct Reference

The Lightweight Thread (LWT) struct.

```
#include <objects.h>
```

### Public Member Functions

- [LIST\\_HEAD](#) (head\_children, [lwt](#)) head\_children
- [LIST\\_ENTRY](#) ([lwt](#)) siblings
- [LIST\\_ENTRY](#) ([lwt](#)) current\_threads
- [TAILQ\\_ENTRY](#) ([lwt](#)) runnable\_threads
- [LIST\\_ENTRY](#) ([lwt](#)) ready\_pool\_threads
- [LIST\\_ENTRY](#) ([lwt](#)) senders
- [TAILQ\\_ENTRY](#) ([lwt](#)) blocked\_senders
- [LIST\\_HEAD](#) (head\_receiver\_channel, [lwt\\_channel](#)) head\_receiver\_channel
- [LIST\\_ENTRY](#) ([lwt](#)) lwts\_in\_kthd

### Data Fields

- long \* [max\\_addr\\_thread\\_stack](#)
- long \* [min\\_addr\\_thread\\_stack](#)
- long \* [thread\\_sp](#)
- [lwt\\_flags\\_t](#) flags
- [lwt\\_t](#) parent
- void \* [sync\\_buffer](#)
- [lwt\\_fnt\\_t](#) start\_routine
- void \* [args](#)
- void \* [return\\_value](#)
- [lwt\\_info\\_t](#) info
- int [id](#)
- [lwt\\_kthd\\_t](#) kthd

### 3.7.1 Detailed Description

The Lightweight Thread (LWT) struct.

## 3.7.2 Member Function Documentation

### 3.7.2.1 lwt\_t::LIST\_ENTRY ( lwt )

Pointers to sibling threads

### 3.7.2.2 lwt\_t::LIST\_ENTRY ( lwt )

Pointers to the current threads

### 3.7.2.3 lwt\_t::LIST\_ENTRY ( lwt )

List of runnable pool threads

### 3.7.2.4 lwt\_t::LIST\_ENTRY ( lwt )

List of senders

### 3.7.2.5 lwt\_t::LIST\_ENTRY ( lwt )

List of lwts in the kthd

### 3.7.2.6 lwt\_t::LIST\_HEAD ( head\_children , lwt )

Head of the list of children lwt's associated with the lwt

### 3.7.2.7 lwt\_t::LIST\_HEAD ( head\_receiver\_channel , lwt\_channel )

Head of the receiver channels associated with the lwt

### 3.7.2.8 lwt\_t::TAILQ\_ENTRY ( lwt )

List of runnable threads

### 3.7.2.9 lwt\_t::TAILQ\_ENTRY ( lwt )

List of blocked senders

## 3.7.3 Field Documentation

### 3.7.3.1 void\* lwt\_t::args

The args for the start\_routine

### 3.7.3.2 `lwt_flags_t lwt_t::flags`

The flags associated with the lwt

### 3.7.3.3 `int lwt_t::id`

The id of the thread

### 3.7.3.4 `lwt_info_t lwt_t::info`

The current status of the thread

### 3.7.3.5 `lwt_kthd_t lwt_t::kthd`

Pointer to kthd

### 3.7.3.6 `long* lwt_t::max_addr_thread_stack`

Pointer to the max address of the stack

### 3.7.3.7 `long* lwt_t::min_addr_thread_stack`

Pointer to the min address of the stack; used for malloc and free

### 3.7.3.8 `lwt_t lwt_t::parent`

Parent thread

### 3.7.3.9 `void* lwt_t::return_value`

The return value from the routine

### 3.7.3.10 `lwt_fnt_t lwt_t::start_routine`

The start routine for the thread to run

### 3.7.3.11 `void* lwt_t::sync_buffer`

Sync buffer

### 3.7.3.12 `long* lwt_t::thread_sp`

The current thread stack pointer for the thread

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

- [objects.h](#)

## 3.8 msort\_args Struct Reference

Struct for passing the args to merge sort around.

### Data Fields

- `int * data`  
*The int array holding randomly generated data.*
- `int * swap`  
*The int array for swap space.*
- `int begin_index`  
*The begin index of the segment.*
- `int end_index`  
*The end index of the segment.*

### 3.8.1 Detailed Description

Struct for passing the args to merge sort around.

### 3.8.2 Field Documentation

#### 3.8.2.1 `int msort_args::begin_index`

The begin index of the segment.

#### 3.8.2.2 `int* msort_args::data`

The int array holding randomly generated data.

#### 3.8.2.3 `int msort_args::end_index`

The end index of the segment.

#### 3.8.2.4 `int* msort_args::swap`

The int array for swap space.

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

- [main\\_chan.c](#)



## Chapter 4

# File Documentation

### 4.1 cas.h File Reference

### 4.2 content.c File Reference

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

#### Macros

- `#define MAX_CONTENT_SZ (1024*1024*10)`

#### Functions

- `char * error_resp (char *path, int *len)`
- `int sanity_check (char *path)`
- `char * content_get (char *path, int *content_len)`

#### 4.2.1 Macro Definition Documentation

4.2.1.1 `#define MAX_CONTENT_SZ (1024*1024*10)`

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

#### 4.2.2 Function Documentation

#### 4.2.2.1 `char* content_get ( char * path, int * content_len )`

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

#### 4.2.2.2 `char* error_resp ( char * path, int * len )`

#### 4.2.2.3 `int sanity_check ( char * path )`

### 4.3 `content.h` File Reference

#### Functions

- `char * content_get (char *path, int *content_len)`

#### 4.3.1 Function Documentation

##### 4.3.1.1 `char* content_get ( char * path, int * content_len )`

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

### 4.4 `enums.h` File Reference

#### Enumerations

- enum `lwt_info_t` {  
`LWT_INFO_NTHD_RUNNABLE`, `LWT_INFO_NTHD_BLOCKED`, `LWT_INFO_NTHD_ZOMBIES`, `LWT_INFO_NTHD_READY_POOL`,  
`LWT_INFO_NCHAN`, `LWT_INFO_NSENDING`, `LWT_INFO_NRECEIVING`, `LWT_INFO_REAPER_READY` }

*The various statuses for a LWT.*

- enum `lwt_flags_t` { `LWT_JOIN` = 0, `LWT_NOJOIN` = 1 }
- enum `lwt_remote_op_t` {  
`LWT_REMOTE_ADD_SENDER_TO_CHANNEL`, `LWT_REMOTE_REMOVE_SENDER_FROM_CHANNEL`, `LWT_REMOTE_ADD_BLOCKED_SENDER_TO_CHANNEL`, `LWT_REMOTE_REMOVE_BLOCKED_SENDER_FROM_CHANNEL`,  
`LWT_REMOTE_ADD_CHANNEL_TO_GROUP`, `LWT_REMOTE_REMOVE_CHANNEL_FROM_GROUP`, `LWT_REMOTE_ADD_EVENT_TO_GROUP`, `LWT_REMOTE_REMOVE_EVENT_FROM_GROUP`,  
`LWT_REMOTE_SIGNAL` }

#### 4.4.1 Enumeration Type Documentation

##### 4.4.1.1 enum `lwt_flags_t`

flags for determining if the lwt is joinable



## Enumerator

**LWT\_JOIN** lwt is joinable  
**LWT\_NOJOIN** lwt is not joinable

## 4.4.1.2 enum lwt\_info\_t

The various statuses for a LWT.

## Enumerator

**LWT\_INFO\_NTHD\_RUNNABLE** Thread state is runnable; it can be switched to  
**LWT\_INFO\_NTHD\_BLOCKED** Thread state is blocked; waiting for another thread to complete  
**LWT\_INFO\_NTHD\_ZOMBIES** Thread state is zombie; thread is dead and needs to be joined  
**LWT\_INFO\_NTHD\_READY\_POOL** Number of ready pool threads  
**LWT\_INFO\_NCHAN** Number of channels that are active  
**LWT\_INFO\_NSENDING** Number of threads blocked sending  
**LWT\_INFO\_NRECEIVING** Number of threads blocked receiving  
**LWT\_INFO\_REAPER\_READY** Reaper is ready to consume

## 4.4.1.3 enum lwt\_remote\_op\_t

## Enumerator

**LWT\_REMOTE\_ADD\_SENDER\_TO\_CHANNEL** Add a lwt sender to a channel  
**LWT\_REMOTE\_REMOVE\_SENDER\_FROM\_CHANNEL** Remove a lwt sender to a channel  
**LWT\_REMOTE\_ADD\_BLOCKED\_SENDER\_TO\_CHANNEL** Add a blocked lwt to a channel  
**LWT\_REMOTE\_REMOVE\_BLOCKED\_SENDER\_FROM\_CHANNEL** Remove a blocked sender from a channel  
**LWT\_REMOTE\_ADD\_CHANNEL\_TO\_GROUP** Add a channel to a group  
**LWT\_REMOTE\_REMOVE\_CHANNEL\_FROM\_GROUP** Remove a channel from the group  
**LWT\_REMOTE\_ADD\_EVENT\_TO\_GROUP** Add an event to remote group  
**LWT\_REMOTE\_REMOVE\_EVENT\_FROM\_GROUP** Remove an event from a remote group  
**LWT\_REMOTE\_SIGNAL** Signal

## 4.5 faa.c File Reference

## Functions

- int [fetch\\_and\\_add](#) (volatile unsigned int \*variable, int value)

*Implementation of fetch and add.*

### 4.5.1 Function Documentation

4.5.1.1 `int fetch_and_add ( volatile unsigned int * variable, int value )` `[inline]`

Implementation of fetch and add.

See also

Taken from wikipedia: <https://www.en.wikipedia.org/wiki/Fetch-and-add>

Parameters

<i>variable</i>	The variable to modify
<i>value</i>	The value to modify

Returns

The updated variable

## 4.6 faa.h File Reference

Functions

- `int fetch_and_add (volatile unsigned int *, int)`

*Implementation of fetch and add.*

### 4.6.1 Function Documentation

4.6.1.1 `int fetch_and_add ( volatile unsigned int * variable, int value )` `[inline]`

Implementation of fetch and add.

See also

Taken from wikipedia: <https://www.en.wikipedia.org/wiki/Fetch-and-add>

Parameters

<i>variable</i>	The variable to modify
<i>value</i>	The value to modify

**Returns**

The updated variable

**4.7 kthd\_server.c File Reference**

```
#include "kthd_server.h"
#include "search.h"
#include "lwt_cgrp.h"
#include "lwt_chan.h"
#include "lwt_kthd.h"
#include "objects.h"
#include "simple_http.h"
#include "content.h"
#include "server.h"
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "assert.h"
#include "string.h"
```

**Macros**

- `#define MAX_CACHE_ENTRIES 10`
- `#define POOL_SIZE 2`
- `#define MAX_ACCEPTORS 2`
- `#define LWT_CACHE 3`
- `#define MAX_REQ_SZ 1024`

**Functions**

- void `respond_and_free_req_kthd` (struct `http_req` \*r, char \*response, int len)
- struct `http_req` \* `newfd_create_req_kthd` (int new\_fd)  
*Helper function for creating an http request.*
- void \* `read_fs` (lwt\_chan\_t cache\_channel)  
*Processes the file system request; used for thread pool.*
- void \* `spawn_fs_workers` (lwt\_chan\_t main\_channel)  
*Wrapper for the file system workers; used for thread pool.*
- void \* `read_cache` (lwt\_chan\_t kthd\_channel)  
*LWT function for caching; checks if the path has been cached; if so, return it; else hit fs threads.*
- void \* `read_cache_kthd` (lwt\_chan\_t main\_channel)  
*Function for running on cache to manage lwt thread pool.*
- void \* `accept_worker` (lwt\_chan\_t main\_channel)  
*Accept worker kthd; accepts the new httpd request.*
- void `process_kthd_server` (int accept\_fd)  
*Main function for the server; sets up channels and then passes data from cache to kthd modules.*

### 4.7.1 Macro Definition Documentation

4.7.1.1 `#define LWT_CACHE 3`

4.7.1.2 `#define MAX_ACCEPTORS 2`

4.7.1.3 `#define MAX_CACHE_ENTRIES 10`

4.7.1.4 `#define MAX_REQ_SZ 1024`

4.7.1.5 `#define POOL_SIZE 2`

### 4.7.2 Function Documentation

4.7.2.1 `void* accept_worker ( lwt_chan_t main_channel )`

Accept worker kthd; accepts the new httpd request.

Parameters

<i>main_channel</i>	The channel to send data across
---------------------	---------------------------------

Returns

NULL

4.7.2.2 `struct http_req* newfd_create_req_kthd ( int new_fd )`

Helper function for creating an http request.

Parameters

<i>new_fd</i>	The file descriptor to open
---------------	-----------------------------

Returns

The Http request received from the file descriptor

4.7.2.3 `void process_kthd_server ( int accept_fd )`

Main function for the server; sets up channels and then passes data from cache to kthd modules.

Parameters

<i>accept_fd</i>	The file descriptor for the http port being used
------------------	--

4.7.2.4 `void* read_cache ( lwt_chan_t kthd_channel )`

LWT function for caching; checks if the path has been cached; if so, return it; else hit fs threads.

## Parameters

<i>kthd_channel</i>	The channel for the spawner
---------------------	-----------------------------

## Returns

NULL

4.7.2.5 void\* read\_cache\_kthd ( lwt\_chan\_t *main\_channel* )

Function for running on cache to manage lwt thread pool.

## Parameters

<i>main_channel</i>	The channel from main used for passing data to other kthds
---------------------	--

## Returns

NULL

4.7.2.6 void\* read\_fs ( lwt\_chan\_t *cache\_channel* )

Processes the file system request; used for thread pool.

## Parameters

<i>cache_channel</i>	The channel to receive
----------------------	------------------------

## Returns

NULL

4.7.2.7 void respond\_and\_free\_req\_kthd ( struct http\_req \* *r*, char \* *response*, int *len* )

4.7.2.8 void\* spawn\_fs\_workers ( lwt\_chan\_t *main\_channel* )

Wrapper for the the file system workers; used for thread pool.

## Parameters

<i>main_channel</i>	The channel for sending the fs channel to
---------------------	---

## 4.8 kthd\_server.h File Reference

### Functions

- void [process\\_kthd\\_server](#) (int accept\_fd)

*Main function for the server; sets up channels and then passes data from cache to kthd modules.*

## 4.8.1 Function Documentation

### 4.8.1.1 `void process_kthd_server ( int accept_fd )`

Main function for the server; sets up channels and then passes data from cache to kthd modules.

## Parameters

<code>accept_fd</code>	The file descriptor for the http port being used
------------------------	--

## 4.9 lwt.c File Reference

```
#include "lwt.h"
#include "lwt_chan.h"
#include "lwt_cgrp.h"
#include "lwt_kthd.h"
#include "pthread.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
```

### Macros

- `#define INIT_ID 1`  
*The initial thread id.*
- `#define DEFAULT_ID -1`  
*The default id provided to threads before actually generating them.*
- `#define POOL_SIZE 100`  
*The size of the pool.*

### Functions

- `void __lwt_dispatch (lwt_t next, lwt_t current)`  
*Dispatch function for switching between threads.*
- `void __lwt_schedule ()`  
*Schedules the next\_current thread to switch to and dispatches.*
- `void __lwt_trampoline ()`  
*Drops in from being scheduled after the initialized thread is switched to and leaps to the function pointer provided.*
- `void * __lwt_stack_get ()`  
*Allocates the stack for a LWT and returns it.*
- `void __lwt_stack_return (void *stack)`  
*Frees the provided stack.*
- `__thread LIST_HEAD (head_current, lwt)`  
*List of all active threads created.*
- `void __insert_runnable_tail (lwt_t thread)`  
*Inserts the given thread to the tail of the runnable thread list.*
- `int lwt_id (lwt_t thread)`  
*Gets the thread id.*
- `lwt_t lwt_current ()`  
*Gets the current thread.*
- `int lwt_info (lwt_info_t t)`  
*Gets the counts of the info.*

- void `__init_lwt_main` (`lwt_t` thread)  
*Initializes the main thread.*
- void `__init_new_lwt` (`lwt_t` thread)  
*Initializes the provided thread.*
- void `__reinit_lwt` (`lwt_t` thread)  
*Reinitializes the given thread.*
- void `__cleanup_joined_thread` (`lwt_t` lwt)  
*Cleans up the thread on join.*
- void \* `lwt_join` (`lwt_t` thread)  
*Joins the provided thread.*
- void `lwt_die` (void \*value)  
*Prepares the current thread to be cleaned up.*
- void `lwt_block` (`lwt_info_t` info)  
*Blocks the current thread.*
- void `lwt_signal` (`lwt_t` thread)  
*Signals the non-running thread to run.*
- int `lwt_yield` (`lwt_t` lwt)  
*Yields to the provided LWT.*
- `__attribute__` ((constructor))  
*Initializes the LWT by wrapping the current thread as a LWT.*
- `__attribute__` ((destructor))  
*Cleans up all remaining threads on exit.*
- `lwt_t` `lwt_create` (`lwt_fnt_t` fn, void \*data, `lwt_flags_t` flags)  
*Creates a LWT using the provided function pointer and the data as input for it.*

## Variables

- `__thread lwt_t` `current_thread` = NULL  
*Pointer to the current thread.*
- `__thread lwt_t` `original_thread` = NULL  
*Pointer to the original/main thread.*

## 4.9.1 Macro Definition Documentation

### 4.9.1.1 `#define DEFAULT_ID -1`

The default id provided to threads before actually generating them.

### 4.9.1.2 `#define INIT_ID 1`

The initial thread id.

### 4.9.1.3 `#define POOL_SIZE 100`

The size of the pool.



## 4.9.2 Function Documentation

### 4.9.2.1 `__attribute__ ( (constructor) )`

Initializes the LWT by wrapping the current thread as a LWT.

### 4.9.2.2 `__attribute__ ( (destructor) )`

Cleans up all remaining threads on exit.

### 4.9.2.3 `void __cleanup_joined_thread ( lwt_t lwt )`

Cleans up the thread on join.

Parameters

<i>lwt</i>	The thread to join on
------------	-----------------------

### 4.9.2.4 `void __init_lwt_main ( lwt_t thread )`

Initializes the main thread.

Parameters

<i>thread</i>	The main thread
---------------	-----------------

### 4.9.2.5 `void __init_new_lwt ( lwt_t thread )`

Initializes the provided thread.

Parameters

<i>thread</i>	The thread to init
---------------	--------------------

### 4.9.2.6 `void __insert_runnable_tail ( lwt_t thread )`

Inserts the given thread to the tail of the runnable thread list.

Parameters

<i>thread</i>	The new thread to be inserted in the list of runnable threads
---------------	---

### 4.9.2.7 `void __lwt_dispatch ( lwt_t next, lwt_t current )`

Dispatch function for switching between threads.

## Parameters

<i>next</i>	The next thread to switch to
<i>current</i>	The current thread

## 4.9.2.8 void \_\_lwt\_schedule ( void )

Schedules the next\_current thread to switch to and dispatches.

## 4.9.2.9 void \* \_\_lwt\_stack\_get ( void )

Allocates the stack for a LWT and returns it.

4.9.2.10 void \_\_lwt\_stack\_return ( void \* *stack* )

Frees the provided stack.

## Parameters

<i>stack</i>	The LWT stack to free
--------------	-----------------------

## 4.9.2.11 void \_\_lwt\_trampoline ( void )

Drops in from being scheduled after the initialized thread is switched to and leaps to the function pointer provided.

4.9.2.12 void \_\_reinit\_lwt ( lwt\_t *thread* )

Reinitializes the given thread.

## Parameters

<i>thread</i>	The thread to reinitialize
---------------	----------------------------

4.9.2.13 \_\_thread LIST\_HEAD ( *head\_current*, lwt )

List of all active threads created.

Counter for the id

## Returns

The next id to use

4.9.2.14 void lwt\_block ( lwt\_info\_t *info* )

Blocks the current thread.

## Parameters

<i>info</i>	The state to set the thread
-------------	-----------------------------

**4.9.2.15** `lwt_t lwt_create ( lwt_fnt_t fn, void * data, lwt_flags_t flags )`

Creates a LWT using the provided function pointer and the data as input for it.

## Parameters

<i>fn</i>	The function pointer to use
<i>data</i>	The data to the function
<i>flags</i>	The flags to be associated with the thread

## Returns

A pointer to the initialized LWT

**4.9.2.16** `lwt_t lwt_current ( ) [inline]`

Gets the current thread.

## Returns

The current thread

**4.9.2.17** `void lwt_die ( void * value )`

Prepares the current thread to be cleaned up.

**4.9.2.18** `int lwt_id ( lwt_t thread ) [inline]`

Gets the thread id.

## Returns

The id of the thread

**4.9.2.19** `int lwt_info ( lwt_info_t t )`

Gets the counts of the info.

## Parameters

<i>t</i>	The info enum to get the counts
----------	---------------------------------

## Returns

The count for the info enum provided

## See also

[lwt\\_info\\_t](#)

#### 4.9.2.20 void\* lwt\_join ( lwt\_t thread )

Joins the provided thread.

##### Parameters

<i>thread</i>	The thread to join on
---------------	-----------------------

#### 4.9.2.21 void lwt\_signal ( lwt\_t thread )

Signals the non-running thread to run.

##### Parameters

<i>thread</i>	The thread to be worken
---------------	-------------------------

#### 4.9.2.22 int lwt\_yield ( lwt\_t lwt )

Yields to the provided LWT.

##### Parameters

<i>lwt</i>	The thread to yield to
------------	------------------------

##### Note

Will just schedule normally if LWT\_NULL is provided

##### Returns

0 if successful

### 4.9.3 Variable Documentation

#### 4.9.3.1 \_\_thread lwt\_t current\_thread = NULL

Pointer to the current thread.

#### 4.9.3.2 \_\_thread lwt\_t original\_thread = NULL

Pointer to the original/main thread.

## 4.10 lwt.h File Reference

```
#include "objects.h"
```

## Functions

- `lwt_t lwt_create (lwt_fnt_t, void *, lwt_flags_t)`  
*Creates a LWT using the provided function pointer and the data as input for it.*
- `void * lwt_join (lwt_t)`  
*Joins the provided thread.*
- `void lwt_die (void *)`  
*Prepares the current thread to be cleaned up.*
- `int lwt_yield (lwt_t)`  
*Yields to the provided LWT.*
- `lwt_t lwt_current ()`  
*Gets the current thread.*
- `int lwt_id (lwt_t)`  
*Gets the thread id.*
- `int lwt_info (lwt_info_t)`  
*Gets the counts of the info.*
- `void lwt_block (lwt_info_t)`  
*Blocks the current thread.*
- `void lwt_signal (lwt_t)`  
*Signals the non-running thread to run.*
- `void __init__ ()`
- `void __destroy__ ()`

### 4.10.1 Function Documentation

4.10.1.1 `void __destroy__ ( )`

4.10.1.2 `void __init__ ( )`

4.10.1.3 `void lwt_block ( lwt_info_t info )`

Blocks the current thread.

Parameters

<i>info</i>	The state to set the thread
-------------	-----------------------------

4.10.1.4 `lwt_t lwt_create ( lwt_fnt_t fn, void * data, lwt_flags_t flags )`

Creates a LWT using the provided function pointer and the data as input for it.

Parameters

<i>fn</i>	The function pointer to use
<i>data</i>	The data to the function
<i>flags</i>	The flags to be associated with the thread

Returns

A pointer to the initialized LWT

#### 4.10.1.5 `lwt_t lwt_current ( )` `[inline]`

Gets the current thread.

##### Returns

The current thread

#### 4.10.1.6 `void lwt_die ( void * )`

Prepares the current thread to be cleaned up.

#### 4.10.1.7 `int lwt_id ( lwt_t thread )` `[inline]`

Gets the thread id.

##### Returns

The id of the thread

#### 4.10.1.8 `int lwt_info ( lwt_info_t t )`

Gets the counts of the info.

##### Parameters

<i>t</i>	The info enum to get the counts
----------	---------------------------------

##### Returns

The count for the info enum provided

##### See also

[lwt\\_info\\_t](#)

#### 4.10.1.9 `void* lwt_join ( lwt_t thread )`

Joins the provided thread.

##### Parameters

<i>thread</i>	The thread to join on
---------------	-----------------------

#### 4.10.1.10 `void lwt_signal ( lwt_t thread )`

Signals the non-running thread to run.

## Parameters

<i>thread</i>	The thread to be worked
---------------	-------------------------

## 4.10.1.11 int lwt\_yield ( lwt\_t lwt )

Yields to the provided LWT.

## Parameters

<i>lwt</i>	The thread to yield to
------------	------------------------

## Note

Will just schedule normally if LWT\_NULL is provided

## Returns

0 if successful

## 4.11 lwt\_cgrp.c File Reference

```
#include "lwt_cgrp.h"
#include "lwt.h"
#include "lwt_chan.h"
#include "stdlib.h"
#include "assert.h"
#include "stdio.h"
#include "sys/queue.h"
```

## Functions

- void [\\_\\_init\\_event](#) (lwt\_chan\_t channel)  
*Initializes the event for when data is added to the channel.*
- void [\\_\\_remove\\_event](#) (lwt\_chan\_t channel, lwt\_cgrp\_t group)  
*Removes an event from the group.*
- lwt\_cgrp\_t [lwt\\_cgrp](#) ()  
*Creates a group of channels.*
- int [lwt\\_cgrp\\_free](#) (lwt\_cgrp\_t group)  
*Frees the group if there are no pending events.*
- int [lwt\\_cgrp\\_add](#) (lwt\_cgrp\_t group, lwt\_chan\_t channel)  
*Adds the channel to the group if the channel hasn't already been added to a group.*
- int [lwt\\_cgrp\\_rem](#) (lwt\_cgrp\_t group, lwt\_chan\_t channel)  
*Removes the channel from the group.*
- lwt\_chan\_t [lwt\\_cgrp\\_wait](#) (lwt\_cgrp\_t group)  
*Waits until there is a pending event in the queue.*
- void [lwt\\_chan\\_mark\\_set](#) (lwt\_chan\_t channel, void \*mark)  
*Marks the channel.*
- void \* [lwt\\_chan\\_mark\\_get](#) (lwt\_chan\_t channel)  
*Grabs the mark from the channel.*

### 4.11.1 Function Documentation

#### 4.11.1.1 void \_\_init\_event ( lwt\_chan\_t channel )

Initializes the event for when data is added to the channel.

##### Parameters

<i>channel</i>	The channel with the new data
<i>sender</i>	The sender lwt

#### 4.11.1.2 void \_\_remove\_event ( lwt\_chan\_t channel, lwt\_cgrp\_t group )

Removes an event from the group.

##### Parameters

<i>channel</i>	The channel (i.e. event) to remove
<i>group</i>	The group to remove the event from

#### 4.11.1.3 lwt\_cgrp\_t lwt\_cgrp ( )

Creates a group of channels.

##### Returns

The group of channels

##### Note

By default, the group is empty

#### 4.11.1.4 int lwt\_cgrp\_add ( lwt\_cgrp\_t group, lwt\_chan\_t channel )

Adds the channel to the group if the channel hasn't already been added to a group.

##### Parameters

<i>group</i>	The group to add the channel to
<i>channel</i>	The channel to add

##### Returns

0 if successful; -1 if the channel is already part of a group

#### 4.11.1.5 int lwt\_cgrp\_free ( lwt\_cgrp\_t group )

Frees the group if there are no pending events.



## Parameters

<i>group</i>	The channel group to free
--------------	---------------------------

## Returns

0 if successful; -1 if there are pending events

4.11.1.6 int lwt\_cgrp\_rem ( lwt\_cgrp\_t *group*, lwt\_chan\_t *channel* )

Removes the channel from the group.

## Parameters

<i>group</i>	The group to remove the channel from
<i>channel</i>	The channel to remove

## Returns

0 if successful; -1 if the channel isn't part of the group; 1 if the group has a pending event

4.11.1.7 lwt\_chan\_t lwt\_cgrp\_wait ( lwt\_cgrp\_t *group* )

Waits until there is a pending event in the queue.

## Parameters

<i>group</i>	The group to wait for
--------------	-----------------------

## Returns

The event in the queue

4.11.1.8 void\* lwt\_chan\_mark\_get ( lwt\_chan\_t *channel* )

Grabs the mark from the channel.

## Parameters

<i>channel</i>	The channel to read
----------------	---------------------

4.11.1.9 void lwt\_chan\_mark\_set ( lwt\_chan\_t *channel*, void \* *mark* )

Marks the channel.

## Parameters

<i>channel</i>	The channel to mark
----------------	---------------------

<i>mark</i>	The marker to set
-------------	-------------------

## 4.12 lwt\_cgrp.h File Reference

```
#include "objects.h"
```

### Functions

- `lwt_cgrp_t lwt_cgrp ()`  
*Creates a group of channels.*
- `int lwt_cgrp_free (lwt_cgrp_t)`  
*Frees the group if there are no pending events.*
- `int lwt_cgrp_add (lwt_cgrp_t, lwt_chan_t)`  
*Adds the channel to the group if the channel hasn't already been added to a group.*
- `int lwt_cgrp_rem (lwt_cgrp_t, lwt_chan_t)`  
*Removes the channel from the group.*
- `lwt_chan_t lwt_cgrp_wait (lwt_cgrp_t)`  
*Waits until there is a pending event in the queue.*
- `void lwt_chan_mark_set (lwt_chan_t, void *)`  
*Marks the channel.*
- `void * lwt_chan_mark_get (lwt_chan_t)`  
*Grabs the mark from the channel.*
- `void __init_event (lwt_chan_t)`  
*Initializes the event for when data is added to the channel.*
- `void __remove_event (lwt_chan_t, lwt_cgrp_t)`  
*Removes an event from the group.*

### 4.12.1 Function Documentation

#### 4.12.1.1 `void __init_event ( lwt_chan_t channel )`

Initializes the event for when data is added to the channel.

##### Parameters

<i>channel</i>	The channel with the new data
<i>sender</i>	The sender lwt

#### 4.12.1.2 `void __remove_event ( lwt_chan_t channel, lwt_cgrp_t group )`

Removes an event from the group.

## Parameters

<i>channel</i>	The channel (i.e. event) to remove
<i>group</i>	The group to remove the event from

## 4.12.1.3 lwt\_cgrp\_t lwt\_cgrp ( )

Creates a group of channels.

## Returns

The group of channels

## Note

By default, the group is empty

## 4.12.1.4 int lwt\_cgrp\_add ( lwt\_cgrp\_t group, lwt\_chan\_t channel )

Adds the channel to the group if the channel hasn't already been added to a group.

## Parameters

<i>group</i>	The group to add the channel to
<i>channel</i>	The channel to add

## Returns

0 if successful; -1 if the channel is already part of a group

## 4.12.1.5 int lwt\_cgrp\_free ( lwt\_cgrp\_t group )

Frees the group if there are no pending events.

## Parameters

<i>group</i>	The channel group to free
--------------	---------------------------

## Returns

0 if successful; -1 if there are pending events

## 4.12.1.6 int lwt\_cgrp\_rem ( lwt\_cgrp\_t group, lwt\_chan\_t channel )

Removes the channel from the group.

**Parameters**

<i>group</i>	The group to remove the channel from
<i>channel</i>	The channel to remove

**Returns**

0 if successful; -1 if the channel isn't part of the group; 1 if the group has a pending event

**4.12.1.7 lwt\_chan\_t lwt\_cgrp\_wait ( lwt\_cgrp\_t *group* )**

Waits until there is a pending event in the queue.

**Parameters**

<i>group</i>	The group to wait for
--------------	-----------------------

**Returns**

The event in the queue

**4.12.1.8 void\* lwt\_chan\_mark\_get ( lwt\_chan\_t *channel* )**

Grabs the mark from the channel.

**Parameters**

<i>channel</i>	The channel to read
----------------	---------------------

**4.12.1.9 void lwt\_chan\_mark\_set ( lwt\_chan\_t *channel*, void \* *mark* )**

Marks the channel.

**Parameters**

<i>channel</i>	The channel to mark
<i>mark</i>	The marker to set

**4.13 lwt\_chan.c File Reference**

```
#include "lwt_chan.h"
#include "lwt.h"
#include "lwt_cgrp.h"
#include "lwt_kthd.h"
#include "objects.h"
#include "stdio.h"
#include "stdlib.h"
#include "assert.h"
#include "faa.h"
```

## Functions

- void [\\_\\_insert\\_sender\\_to\\_chan](#) (lwt\_chan\_t chan, lwt\_t lwt)  
*Inserts the sender into the channel.*
- void [\\_\\_remove\\_sender\\_from\\_chan](#) (lwt\_chan\_t chan, lwt\_t lwt)  
*Removes the sender from the channel.*
- void [\\_\\_insert\\_blocked\\_sender\\_to\\_chan](#) (lwt\_chan\_t chan, lwt\_t lwt)  
*Inserts the sender onto the blocked queue.*
- void [\\_\\_remove\\_blocked\\_sender\\_from\\_chan](#) (lwt\_chan\_t chan, lwt\_t lwt)  
*Removes the sender from the channel's blocked queue.*
- void \* [\\_\\_pop\\_data\\_from\\_async\\_buffer](#) (lwt\_chan\_t c)  
*Pops the data into the buffer.*
- lwt\_chan\_t [lwt\\_chan](#) (int sz)  
*Creates the channel on the receiving thread.*
- int [lwt\\_snd](#) (lwt\_chan\_t c, void \*data)  
*Sends the data over the channel to the receiver.*
- int [lwt\\_snd\\_chan](#) (lwt\_chan\_t c, lwt\_chan\_t sending)  
*Sends sending over the channel c.*
- lwt\_chan\_t [lwt\\_rcv\\_chan](#) (lwt\_chan\_t c)  
*Receives the data over the channel.*
- void [lwt\\_chan\\_deref](#) (lwt\_chan\_t c)  
*Deallocates the channel only if no threads still have references to the channel.*
- void \* [lwt\\_rcv](#) (lwt\_chan\_t c)  
*Receives the data from the channel and returns it.*
- lwt\_t [lwt\\_create\\_chan](#) (lwt\_chan\_fn\_t fn, lwt\_chan\_t c, lwt\_flags\_t flags)  
*Creates a lwt with the channel as an arg.*

### 4.13.1 Function Documentation

#### 4.13.1.1 void \_\_insert\_blocked\_sender\_to\_chan ( lwt\_chan\_t chan, lwt\_t lwt )

Inserts the sender onto the blocked queue.

##### Parameters

<i>chan</i>	The channel owning the queue
<i>lwt</i>	The sender to add to the queue

#### 4.13.1.2 void \_\_insert\_sender\_to\_chan ( lwt\_chan\_t chan, lwt\_t lwt )

Inserts the sender into the channel.

##### Parameters

<i>chan</i>	The channel to insert the sender
-------------	----------------------------------

<i>lwt</i>	The sender lwt
------------	----------------

#### 4.13.1.3 void\* \_\_pop\_data\_from\_async\_buffer ( lwt\_chan\_t c )

Pops the data into the buffer.

##### Parameters

<i>c</i>	The channel to remove the data from
<i>data</i>	The data to remove If the buffer is empty, it will block until there is something to read

#### 4.13.1.4 void \_\_remove\_blocked\_sender\_from\_chan ( lwt\_chan\_t chan, lwt\_t lwt )

Removes the sender from the channel's blocked queue.

##### Parameters

<i>chan</i>	The channel owning the queue
<i>lwt</i>	The sender to remove from the queue

#### 4.13.1.5 void \_\_remove\_sender\_from\_chan ( lwt\_chan\_t chan, lwt\_t lwt )

Removes the sender from the channel.

##### Parameters

<i>chan</i>	The channel to remove the sender from
<i>lwt</i>	The sender to remove

#### 4.13.1.6 lwt\_chan\_t lwt\_chan ( int sz )

Creates the channel on the receiving thread.

##### Parameters

<i>sz</i>	The size of the buffer
-----------	------------------------

##### Returns

A pointer to the initialized channel

#### 4.13.1.7 void lwt\_chan\_deref ( lwt\_chan\_t c )

Deallocates the channel only if no threads still have references to the channel.

##### Parameters

---

<i>c</i>	The channel to deallocate
----------	---------------------------

#### 4.13.1.8 lwt\_t lwt\_create\_chan ( lwt\_chan\_fn\_t fn, lwt\_chan\_t c, lwt\_flags\_t flags )

Creates a lwt with the channel as an arg.

##### Parameters

<i>fn</i>	The function to use to create the thread
<i>c</i>	The channel to send
<i>flags</i>	The flags for the thread

##### Returns

The thread to return

#### 4.13.1.9 void\* lwt\_rcv ( lwt\_chan\_t c )

Receives the data from the channel and returns it.

##### Parameters

<i>c</i>	The channel to receive from
----------	-----------------------------

##### Returns

The data from the channel

#### 4.13.1.10 lwt\_chan\_t lwt\_rcv\_chan ( lwt\_chan\_t c )

Receives the data over the channel.

##### Parameters

<i>c</i>	The channel to use for receiving
----------	----------------------------------

##### Returns

The channel being sent over c

#### 4.13.1.11 int lwt\_snd ( lwt\_chan\_t c, void \* data )

Sends the data over the channel to the receiver.

##### Parameters

<i>c</i>	The channel to use for sending
----------	--------------------------------

<i>data</i>	The data for sending
-------------	----------------------

#### Returns

-1 if there is no receiver; 0 if successful

#### 4.13.1.12 `int lwt_snd_chan ( lwt_chan_t c, lwt_chan_t sending )`

Sends sending over the channel c.

#### Parameters

<i>c</i>	The channel to send sending across
<i>sending</i>	The channel to send

## 4.14 `lwt_chan.h` File Reference

```
#include "objects.h"
```

#### Functions

- `lwt_chan_t lwt_chan` (int)  
*Creates the channel on the receiving thread.*
- `void lwt_chan_deref` (lwt\_chan\_t)  
*Deallocates the channel only if no threads still have references to the channel.*
- `int lwt_snd` (lwt\_chan\_t, void \*)  
*Sends the data over the channel to the receiver.*
- `void * lwt_rcv` (lwt\_chan\_t)  
*Receives the data from the channel and returns it.*
- `int lwt_snd_chan` (lwt\_chan\_t, lwt\_chan\_t)  
*Sends sending over the channel c.*
- `lwt_chan_t lwt_rcv_chan` (lwt\_chan\_t)  
*Receives the data over the channel.*
- `lwt_t lwt_create_chan` (lwt\_chan\_fn\_t, lwt\_chan\_t, lwt\_flags\_t)  
*Creates a lwt with the channel as an arg.*
- `void __insert_sender_to_chan` (lwt\_chan\_t, lwt\_t)  
*Inserts the sender into the channel.*
- `void __remove_sender_from_chan` (lwt\_chan\_t, lwt\_t)  
*Removes the sender from the channel.*
- `void __insert_blocked_sender_to_chan` (lwt\_chan\_t, lwt\_t)  
*Inserts the sender onto the blocked queue.*
- `void __remove_blocked_sender_from_chan` (lwt\_chan\_t, lwt\_t)  
*Removes the sender from the channel's blocked queue.*



### 4.14.1 Function Documentation

4.14.1.1 void \_\_insert\_blocked\_sender\_to\_chan ( lwt\_chan\_t *chan*, lwt\_t *lwt* )

Inserts the sender onto the blocked queue.

## Parameters

<i>chan</i>	The channel owning the queue
<i>lwt</i>	The sender to add to the queue

4.14.1.2 void \_\_insert\_sender\_to\_chan ( lwt\_chan\_t *chan*, lwt\_t *lwt* )

Inserts the sender into the channel.

## Parameters

<i>chan</i>	The channel to insert the sender
<i>lwt</i>	The sender lwt

4.14.1.3 void \_\_remove\_blocked\_sender\_from\_chan ( lwt\_chan\_t *chan*, lwt\_t *lwt* )

Removes the sender from the channel's blocked queue.

## Parameters

<i>chan</i>	The channel owning the queue
<i>lwt</i>	The sender to remove from the queue

4.14.1.4 void \_\_remove\_sender\_from\_chan ( lwt\_chan\_t *chan*, lwt\_t *lwt* )

Removes the sender from the channel.

## Parameters

<i>chan</i>	The channel to remove the sender from
<i>lwt</i>	The sender to remove

4.14.1.5 lwt\_chan\_t lwt\_chan ( int *sz* )

Creates the channel on the receiving thread.

## Parameters

<i>sz</i>	The size of the buffer
-----------	------------------------

## Returns

A pointer to the initialized channel

4.14.1.6 void lwt\_chan\_deref ( lwt\_chan\_t *c* )

Deallocates the channel only if no threads still have references to the channel.

## Parameters

<i>c</i>	The channel to deallocate
----------	---------------------------

## 4.14.1.7 lwt\_t lwt\_create\_chan ( lwt\_chan\_fn\_t fn, lwt\_chan\_t c, lwt\_flags\_t flags )

Creates a lwt with the channel as an arg.

## Parameters

<i>fn</i>	The function to use to create the thread
<i>c</i>	The channel to send
<i>flags</i>	The flags for the thread

## Returns

The thread to return

## 4.14.1.8 void\* lwt\_rcv ( lwt\_chan\_t c )

Receives the data from the channel and returns it.

## Parameters

<i>c</i>	The channel to receive from
----------	-----------------------------

## Returns

The data from the channel

## 4.14.1.9 lwt\_chan\_t lwt\_rcv\_chan ( lwt\_chan\_t c )

Receives the data over the channel.

## Parameters

<i>c</i>	The channel to use for receiving
----------	----------------------------------

## Returns

The channel being sent over c

## 4.14.1.10 int lwt\_snd ( lwt\_chan\_t c, void \* data )

Sends the data over the channel to the receiver.

## Parameters

<code>c</code>	The channel to use for sending
<code>data</code>	The data for sending

#### Returns

-1 if there is no receiver; 0 if successful

#### 4.14.1.11 `int lwt_snd_chan ( lwt_chan_t c, lwt_chan_t sending )`

Sends sending over the channel c.

#### Parameters

<code>c</code>	The channel to send sending across
<code>sending</code>	The channel to send

## 4.15 `lwt_kthd.c` File Reference

```
#include "lwt_kthd.h"
#include "lwt.h"
#include "lwt_chan.h"
#include "assert.h"
#include "pthread.h"
#include "faa.h"
#include "stdio.h"
```

### Functions

- void \* [pthread\\_function](#) (void \*data)  
*Function for the kthd (i.e. pthread) LWT wrapper to perform.*
- int [lwt\\_kthd\\_create](#) (lwt\_chan\_fn\_t fn, lwt\_chan\_t c, lwt\_flags\_t flags)  
*Creates an N:M kthd.*
- struct kthd\_event \* [\\_\\_pop\\_from\\_buffer](#) (lwt\_kthd\_t kthd)  
*Pops a kthd event from the buffer.*
- int [\\_\\_push\\_to\\_buffer](#) (lwt\_kthd\_t kthd, struct kthd\_event \*data)  
*Pushes a kthd event into the event buffer.*
- void [\\_\\_init\\_kthd](#) (lwt\_t lwt)  
*Initializes a kthd.*
- void \* [\\_\\_lwt\\_buffer](#) (void \*d)  
*Function for the reaper lwt; when all other lwts are blocked, processes events for the kthd.*
- lwt\_kthd\_t [\\_\\_get\\_kthd](#) ()  
*Helper method for returning the current kthd.*
- void [\\_\\_init\\_kthd\\_event](#) (lwt\_t remote\_lwt, lwt\_chan\_t remote\_chan, lwt\_cgrp\_t remote\_group, lwt\_kthd\_t kthd, lwt\_remote\_op\_t remote\_op, int block)  
*Initializes a kthd event.*

## Variables

- `__thread lwt_kthd_t pthread_kthd`  
Pointer to the kthd for the pthread.

### 4.15.1 Function Documentation

#### 4.15.1.1 `lwt_kthd_t __get_kthd ( )`

Helper method for returning the current kthd.

##### Returns

The current kthd

#### 4.15.1.2 `void __init_kthd ( lwt_t lwt )`

Initializes a kthd.

##### Parameters

<i>lwt</i>	The lwt for the kthd
------------	----------------------

#### 4.15.1.3 `void __init_kthd_event ( lwt_t remote_lwt, lwt_chan_t remote_chan, lwt_cgrp_t remote_group, lwt_kthd_t kthd, lwt_remote_op_t remote_op, int block )`

Initializes a kthd event.

##### Parameters

<i>remote_lwt</i>	The lwt to modify
<i>remote_chan</i>	The channel to modify
<i>remote_group</i>	The group to modify
<i>kthd</i>	The kthd to modify
<i>remote_op</i>	The operation to perform
<i>block</i>	Is the operation blocking (generally yes; signal is not)

#### 4.15.1.4 `void* __lwt_buffer ( void * d )`

Function for the reaper lwt; when all other lwts are blocked, processes events for the kthd.

##### Parameters

<i>d</i>	Data; unused; needed to match file signature
----------	--

##### Returns

NULL

#### 4.15.1.5 `struct kthd_event* __pop_from_buffer ( lwt_kthd_t kthd )`

Pops a kthd event from the buffer.

**Parameters**

<i>kthd</i>	The kthd to pop
-------------	-----------------

**Returns**

The kthd event for the action to perform in the reaper function

#### 4.15.1.6 `int __push_to_buffer ( lwt_kthd_t kthd, struct kthd_event * data )`

Pushes a kthd event into the event buffer.

**Parameters**

<i>kthd</i>	The kthd to modify
<i>data</i>	The data to insert

**Returns**

0 if successful; -1 if not

#### 4.15.1.7 `int lwt_kthd_create ( lwt_chan_fn_t fn, lwt_chan_t c, lwt_flags_t flags )`

Creates an N:M kthd.

**Parameters**

<i>fn</i>	The channel function to run on the remote kthd
<i>c</i>	The channel used as input to that function
<i>flags</i>	The flags for the function

#### 4.15.1.8 `void* pthread_function ( void * data )`

Function for the kthd (i.e. pthread) LWT wrapper to perform.

**Parameters**

<i>data</i>	The kthd data used for storing the params for the create chan call
-------------	--

**Returns**

NULL

### 4.15.2 Variable Documentation

#### 4.15.2.1 `__thread lwt_kthd_t pthread_kthd`

Pointer to the kthd for the pthread.

## 4.16 lwt\_kthd.h File Reference

```
#include "objects.h"
```

### Functions

- `int lwt_kthd_create (lwt_chan_fn_t, lwt_chan_t, lwt_flags_t)`  
*Creates an N:M kthd.*
- `void __init_kthd (lwt_t)`  
*Initializes a kthd.*
- `void __insert_lwt_into_tail (lwt_kthd_t, lwt_t)`
- `void __remove_lwt_from_kthd (lwt_kthd_t, lwt_t)`
- `lwt_kthd_t __get_kthd ()`  
*Helper method for returning the current kthd.*
- `void * __lwt_buffer (void *)`  
*Function for the reaper lwt; when all other lwts are blocked, processes events for the kthd.*
- `void __init_kthd_event (lwt_t, lwt_chan_t, lwt_cgrp_t, lwt_kthd_t, lwt_remote_op_t, int)`  
*Initializes a kthd event.*

### 4.16.1 Function Documentation

#### 4.16.1.1 lwt\_kthd\_t \_\_get\_kthd ( )

Helper method for returning the current kthd.

#### Returns

The current kthd

#### 4.16.1.2 void \_\_init\_kthd ( lwt\_t lwt )

Initializes a kthd.

#### Parameters

<i>lwt</i>	The lwt for the kthd
------------	----------------------

#### 4.16.1.3 void \_\_init\_kthd\_event ( lwt\_t remote\_lwt, lwt\_chan\_t remote\_chan, lwt\_cgrp\_t remote\_group, lwt\_kthd\_t kthd, lwt\_remote\_op\_t remote\_op, int block )

Initializes a kthd event.

#### Parameters

<i>remote_lwt</i>	The lwt to modify
<i>remote_chan</i>	The channel to modify
<i>remote_group</i>	The group to modify
<i>kthd</i>	The kthd to modify
<i>remote_op</i>	The operation to perform
<i>block</i>	Is the operation blocking (generally yes; signal is not)

4.16.1.4 `void __insert_lwt_into_tail ( lwt_kthd_t, lwt_t )`

4.16.1.5 `void* __lwt_buffer ( void * d )`

Function for the reaper lwt; when all other lwts are blocked, processes events for the kthd.

#### Parameters

<i>d</i>	Data; unused; needed to match file signature
----------	--

#### Returns

NULL

4.16.1.6 `void __remove_lwt_from_kthd ( lwt_kthd_t, lwt_t )`

4.16.1.7 `int lwt_kthd_create ( lwt_chan_fn_t fn, lwt_chan_t c, lwt_flags_t flags )`

Creates an N:M kthd.

#### Parameters

<i>fn</i>	The channel function to run on the remote kthd
<i>c</i>	The channel used as input to that function
<i>flags</i>	The flags for the function

## 4.17 main.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <sys/wait.h>
#include <pthread.h>
#include "util.h"
#include "server.h"
#include "kthd_server.h"
#include "cas.h"
```

#### Macros

- `#define MAX_DATA_SZ 1024`



- #define `MAX_CONCURRENCY` 4
- #define `BUFFER_LENGTH` 256

## Enumerations

- enum `server_type_t` { `SERVER_TYPE_ONE` = 0, `SERVER_TYPE_TWO` = 1 }

## Functions

- void `server_single_request` (int `accept_fd`)
- int `main` (int `argc`, char \*`argv`[])

### 4.17.1 Macro Definition Documentation

4.17.1.1 #define `BUFFER_LENGTH` 256

4.17.1.2 #define `MAX_CONCURRENCY` 4

4.17.1.3 #define `MAX_DATA_SZ` 1024

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

### 4.17.2 Enumeration Type Documentation

4.17.2.1 enum `server_type_t`

Enumerator

**`SERVER_TYPE_ONE`**

**`SERVER_TYPE_TWO`**

### 4.17.3 Function Documentation

4.17.3.1 int `main` ( int *argc*, char \* *argv*[] )

4.17.3.2 void `server_single_request` ( int *accept\_fd* )

## 4.18 main3.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "lwt.h"
#include "lwt_chan.h"
#include "lwt_cgrp.h"
```

## Macros

- `#define rdtsc1(val) __asm__ __volatile__("rdtsc" : "=A" (val))`
- `#define ITER 10000`
- `#define IS_RESET()`
- `#define GRPSZ 3`

## Functions

- `void * fn_bounce (void *d)`
- `void * fn_null (void *d)`
- `void test_perf (void)`
- `void * fn_identity (void *d)`
- `void * fn_nested_joins (void *d)`
- `void * fn_sequence (void *d)`
- `void * fn_join (void *d)`
- `void test_crt_join_sched (void)`
- `void * fn_chan (lwt_chan_t to)`
- `void test_perf_channels (int chsz)`
- `void * fn_snder (lwt_chan_t c, int v)`
- `void * fn_snder_1 (lwt_chan_t c)`
- `void * fn_snder_2 (lwt_chan_t c)`
- `void test_multisend (int chsz)`
- `void * fn_async_steam (lwt_chan_t to)`
- `void test_perf_async_steam (int chsz)`
- `void * fn_grpwait (lwt_chan_t c)`
- `void test_grpwait (int chsz, int grpsz)`
- `int main (void)`

## Variables

- `volatile int sched [2] = {0, 0}`
- `volatile int curr = 0`

### 4.18.1 Macro Definition Documentation

#### 4.18.1.1 `#define GRPSZ 3`

#### 4.18.1.2 `#define IS_RESET( )`

#### Value:

```
assert( lwt_info(LWT_INFO_NTHD_RUNNABLE) == 1 && \
        lwt_info(LWT_INFO_NTHD_ZOMBIES) == 0 && \
        lwt_info(LWT_INFO_NTHD_BLOCKED) == 0)
```

4.18.1.3 `#define ITER 10000`

4.18.1.4 `#define rdtscll( val ) __asm__ __volatile__( "rdtsc" : "=A" (val))`

## 4.18.2 Function Documentation

4.18.2.1 `void* fn_async_steam ( lwt_chan_t to )`

4.18.2.2 `void* fn_bounce ( void * d )`

4.18.2.3 `void* fn_chan ( lwt_chan_t to )`

4.18.2.4 `void* fn_grpwait ( lwt_chan_t c )`

4.18.2.5 `void* fn_identity ( void * d )`

4.18.2.6 `void* fn_join ( void * d )`

4.18.2.7 `void* fn_nested_joins ( void * d )`

4.18.2.8 `void* fn_null ( void * d )`

4.18.2.9 `void* fn_sequence ( void * d )`

4.18.2.10 `void* fn_snder ( lwt_chan_t c, int v )`

4.18.2.11 `void* fn_snder_1 ( lwt_chan_t c )`

4.18.2.12 `void* fn_snder_2 ( lwt_chan_t c )`

4.18.2.13 `int main ( void )`

4.18.2.14 `void test_crt_join_sched ( void )`

4.18.2.15 `void test_grpwait ( int chsz, int grpsz )`

Q: why don't we iterate through all of the data here?

A: We need to fix 1) `cevt_wait` to be level triggered, or 2) provide a function to detect if there is data available on a channel. Either of these would allow us to iterate on a channel while there is more data pending.

4.18.2.16 `void test_multisend ( int chsz )`

4.18.2.17 `void test_perf ( void )`

4.18.2.18 `void test_perf_async_steam ( int chsz )`

4.18.2.19 `void test_perf_channels ( int chsz )`

## 4.18.3 Variable Documentation

4.18.3.1 `volatile int curr = 0`

4.18.3.2 `volatile int sched[2] = {0, 0}`

## 4.19 `main_chan.c` File Reference

```
#include "lwt.h"
#include "lwt_chan.h"
#include "lwt_cgrp.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
```

### Data Structures

- struct [msort\\_args](#)  
*Struct for passing the args to merge sort around.*

### Macros

- `#define ITER 80`
- `#define MERGE_SZ 80`

### Functions

- void \* [msort](#) (lwt\_chan\_t main\_channel)  
*Merge sort in parallel.*
- void [merge\\_sort\\_test](#) ()  
*Runs the merge sort test Tests being able to create multiple child channels and joining them properly.*
- void \* [child\\_ping](#) (lwt\_chan\_t main\_channel)  
*Ping channel test.*
- void \* [child\\_pong](#) (lwt\_chan\_t main\_channel)  
*Receives a count, updates it and sends it back.*
- void [ping\\_pong\\_test](#) ()  
*Runs the ping/pong test.*
- void \* [child\\_multiple\\_channels](#) (lwt\_chan\_t main\_channel)
- void [multiple\\_channels\\_test](#) ()
- void [multiple\\_channels\\_test\\_v2](#) ()
- void [multiple\\_channels\\_test\\_v3](#) ()
- int [main](#) ()

### 4.19.1 Macro Definition Documentation

4.19.1.1 `#define ITER 80`

4.19.1.2 `#define MERGE_SZ 80`

## 4.19.2 Function Documentation

4.19.2.1 void\* child\_multiple\_channels ( lwt\_chan\_t *main\_channel* )

4.19.2.2 void\* child\_ping ( lwt\_chan\_t *main\_channel* )

Ping channel test.

### Parameters

<i>main_channel</i>	The channel from the main thread
---------------------	----------------------------------

### Returns

0 if successful Sends count out to many siblings; tests that they receive and update it properly

4.19.2.3 void\* child\_pong ( lwt\_chan\_t *main\_channel* )

Receives a count, updates it and sends it back.

### Parameters

<i>main_channel</i>	The channel from the main thread
---------------------	----------------------------------

### Returns

0 if successful

4.19.2.4 int main ( void )

Main function

4.19.2.5 void merge\_sort\_test ( )

Runs the merge sort test Tests being able to create multiple child channels and joining them properly.

4.19.2.6 void\* msort ( lwt\_chan\_t *main\_channel* )

Merge sort in parallel.

### Parameters

<i>main_channel</i>	The channel from the main thread
---------------------	----------------------------------

### Returns

0 if successful

### Note

Adapted from wikipedia: [http://en.wikipedia.org/wiki/Merge\\_sort#Parallel\\_merge\\_sort](http://en.wikipedia.org/wiki/Merge_sort#Parallel_merge_sort)

4.19.2.7 void multiple\_channels\_test ( )

4.19.2.8 void multiple\_channels\_test\_v2 ( )

4.19.2.9 void multiple\_channels\_test\_v3 ( )

4.19.2.10 void ping\_pong\_test ( )

Runs the ping/pong test.

## 4.20 main\_kthd.c File Reference

```
#include "lwt_kthd.h"
#include "lwt_chan.h"
#include "lwt.h"
#include "stdio.h"
#include "assert.h"
```

### Macros

- #define [MAX\\_PING\\_PONG\\_VALUE](#) 100
- #define [ITER](#) 10000
- #define [GRPSZ](#) 3

### Functions

- void \* [kthd\\_ping](#) (lwt\_chan\_t ping\_channel)
- void [kthd\\_ping\\_pong\\_sync](#) ()
- void \* [fn\\_grpwait](#) (lwt\_chan\_t c)
- void [test\\_grpwait](#) (int chsz, int grpsz)
- int [main](#) ()

### 4.20.1 Macro Definition Documentation

4.20.1.1 #define [GRPSZ](#) 3

4.20.1.2 #define [ITER](#) 10000

4.20.1.3 #define [MAX\\_PING\\_PONG\\_VALUE](#) 100

### 4.20.2 Function Documentation

4.20.2.1 void\* [fn\\_grpwait](#) ( lwt\_chan\_t c )

4.20.2.2 void\* [kthd\\_ping](#) ( lwt\_chan\_t ping\_channel )

4.20.2.3 void [kthd\\_ping\\_pong\\_sync](#) ( )

4.20.2.4 `int main ( void )`

4.20.2.5 `void test_grpwait ( int chsz, int grpsz )`

Q: why don't we iterate through all of the data here?

A: We need to fix 1) `cevt_wait` to be level triggered, or 2) provide a function to detect if there is data available on a channel. Either of these would allow us to iterate on a channel while there is more data pending.

## 4.21 main\_orig.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "lwt.h"
```

### Macros

- `#define rdtsc1(val) __asm__ __volatile__ ("rdtsc" : "=A" (val))`
- `#define ITER 10000`
- `#define IS_RESET()`

### Functions

- `void * fn_bounce (void *d)`
- `void * fn_null (void *d)`
- `void test_perf (void)`
- `void * fn_identity (void *d)`
- `void * fn_nested_joins (void *d)`
- `void * fn_sequence (void *d)`
- `void * fn_join (void *d)`
- `void test_crt_join_sched (void)`
- `int main (void)`

### Variables

- `volatile int sched [2] = {0, 0}`
- `volatile int curr = 0`

### 4.21.1 Macro Definition Documentation

#### 4.21.1.1 `#define IS_RESET( )`

**Value:**

```
assert ( lwt_info(LWT_INFO_NTHD_RUNNABLE) == 1 && \
         lwt_info(LWT_INFO_NTHD_ZOMBIES) == 0 && \
         lwt_info(LWT_INFO_NTHD_BLOCKED) == 0)
```

4.21.1.2 `#define ITER 10000`

4.21.1.3 `#define rdtsc( val ) __asm__ __volatile__( "rdtsc" : "=A" (val))`

## 4.21.2 Function Documentation

4.21.2.1 `void* fn_bounce ( void * d )`

4.21.2.2 `void* fn_identity ( void * d )`

4.21.2.3 `void* fn_join ( void * d )`

4.21.2.4 `void* fn_nested_joins ( void * d )`

4.21.2.5 `void* fn_null ( void * d )`

4.21.2.6 `void* fn_sequence ( void * d )`

4.21.2.7 `int main ( void )`

4.21.2.8 `void test_crt_join_sched ( void )`

4.21.2.9 `void test_perf ( void )`

## 4.21.3 Variable Documentation

4.21.3.1 `volatile int curr = 0`

4.21.3.2 `volatile int sched[2] = {0, 0}`

## 4.22 objects.h File Reference

```
#include "pthread.h"
#include "stdlib.h"
#include <sys/queue.h>
#include "enums.h"
```

### Data Structures

- struct [lwt\\_cgrp\\_t](#)  
*Channel group for handling events within a group.*
- struct [lwt\\_chan\\_t](#)  
*The channel for synchronous and asynchronous communication.*
- struct [kthd\\_event](#)
- struct [lwt\\_kthd\\_t](#)
- struct [lwt\\_kthd\\_data](#)
- struct [lwt\\_t](#)  
*The Lightweight Thread (LWT) struct.*



## Macros

- `#define EVENT_BUFFER_SIZE 10000`
- `#define PAGE_SIZE 4096`
- `#define NUM_PAGES 5`
- `#define STACK_SIZE PAGE_SIZE*NUM_PAGES`
- `#define DEBUG 1`
- `#define LWT_NULL NULL`

## Typedefs

- `typedef void (*)(lwt_chan_fn_t)(lwt_chan_t)`
- `typedef void (*)(lwt_fnt_t)(void *)`

### 4.22.1 Macro Definition Documentation

#### 4.22.1.1 `#define DEBUG 1`

#### 4.22.1.2 `#define EVENT_BUFFER_SIZE 10000`

Size of the event buffer

#### 4.22.1.3 `#define LWT_NULL NULL`

Null id for yields

#### 4.22.1.4 `#define NUM_PAGES 5`

Number of pages to allocate to the stack

#### 4.22.1.5 `#define PAGE_SIZE 4096`

Size of the a page in the OS -> 4K

#### 4.22.1.6 `#define STACK_SIZE PAGE_SIZE*NUM_PAGES`

Size of the stack

### 4.22.2 Typedef Documentation

#### 4.22.2.1 `typedef void (*)(lwt_chan_fn_t)(lwt_chan_t)`

#### 4.22.2.2 `typedef void (*)(lwt_fnt_t)(void *)`

## 4.23 server.c File Reference

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <errno.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
```

### Functions

- int [server\\_create](#) (short int port)
- int [server\\_accept](#) (int fd)

#### 4.23.1 Function Documentation

4.23.1.1 int [server\\_accept](#) ( int *fd* )

4.23.1.2 int [server\\_create](#) ( short int *port* )

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

## 4.24 server.h File Reference

### Functions

- int [server\\_create](#) (short int port)
- int [server\\_accept](#) (int fd)

#### 4.24.1 Function Documentation

4.24.1.1 int [server\\_accept](#) ( int *fd* )

4.24.1.2 int [server\\_create](#) ( short int *port* )

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

## 4.25 simple\_http.c File Reference

```
#include <string.h>
```

```
#include <assert.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include "simple_http.h"
```

## Macros

- `#define MAX_DIGITS 128`

## Functions

- `struct http_req * shttp_alloc_req` (int fd, char \*request)
- `void shttp_free_req` (struct http\_req \*r)
- `int shttp_get_path` (struct http\_req \*r)
- `int shttp_alloc_response_head` (struct http\_req \*r, char \*data, int dlen)

### 4.25.1 Macro Definition Documentation

4.25.1.1 `#define MAX_DIGITS 128`

### 4.25.2 Function Documentation

4.25.2.1 `struct http_req* shttp_alloc_req ( int fd, char * request )`

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

4.25.2.2 `int shttp_alloc_response_head ( struct http_req * r, char * data, int dlen )`

4.25.2.3 `void shttp_free_req ( struct http_req * r )`

4.25.2.4 `int shttp_get_path ( struct http_req * r )`

## 4.26 simple\_http.h File Reference

### Data Structures

- `struct http_req`

### Functions

- `struct http_req * shttp_alloc_req` (int fd, char \*request)
- `void shttp_free_req` (struct http\_req \*r)
- `int shttp_get_path` (struct http\_req \*r)
- `int shttp_alloc_response_head` (struct http\_req \*r, char \*resp, int rlen)

### 4.26.1 Function Documentation

4.26.1.1 `struct http_req* shttp_alloc_req ( int fd, char * request )`

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

4.26.1.2 `int shttp_alloc_response_head ( struct http_req * r, char * resp, int rlen )`

4.26.1.3 `void shttp_free_req ( struct http_req * r )`

4.26.1.4 `int shttp_get_path ( struct http_req * r )`

## 4.27 util.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include "server.h"
#include "simple_http.h"
#include "content.h"
```

### Macros

- `#define MAX_REQ_SZ 1024`

### Functions

- `struct http_req * newfd_create_req (int new_fd)`
- `void respond_and_free_req (struct http_req *r, char *response, int len)`
- `void client_process (int fd)`

### 4.27.1 Macro Definition Documentation

4.27.1.1 `#define MAX_REQ_SZ 1024`

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

### 4.27.2 Function Documentation

4.27.2.1 `void client_process ( int fd )`

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

4.27.2.2 `struct http_req* newfd_create_req ( int new_fd )`

4.27.2.3 `void respond_and_free_req ( struct http_req * r, char * response, int len )`

## 4.28 util.h File Reference

### Functions

- void `client_process` (int *fd*)

### 4.28.1 Function Documentation

4.28.1.1 `void client_process ( int fd )`

Redistribution of this file is permitted under the GNU General Public License v2.

Copyright 2012 by Gabriel Parmer. Author: Gabriel Parmer, [gparmer@gwu.edu](mailto:gparmer@gwu.edu), 2012

# Index

## args

lwt, [13](#)

## enums.h

LWT\_INFO\_NCHAN, [19](#)  
LWT\_INFO\_NRECEIVING, [19](#)  
LWT\_INFO\_NSENDING, [19](#)  
LWT\_INFO\_NTHD\_BLOCKED, [19](#)  
LWT\_INFO\_NTHD\_READY\_POOL, [19](#)  
LWT\_INFO\_NTHD\_RUNNABLE, [19](#)  
LWT\_INFO\_NTHD\_ZOMBIES, [19](#)  
LWT\_INFO\_REAPER\_READY, [19](#)  
LWT\_JOIN, [19](#)  
LWT\_NOJOIN, [19](#)  
LWT\_REMOTE\_ADD\_BLOCKED\_SENDER\_TO\_↔  
CHANNEL, [19](#)  
LWT\_REMOTE\_ADD\_CHANNEL\_TO\_GROUP, [19](#)  
LWT\_REMOTE\_ADD\_EVENT\_TO\_GROUP, [19](#)  
LWT\_REMOTE\_ADD\_SENDER\_TO\_CHANNEL, [19](#)  
LWT\_REMOTE\_REMOVE\_BLOCKED\_SENDER\_↔  
FROM\_CHANNEL, [19](#)  
LWT\_REMOTE\_REMOVE\_CHANNEL\_FROM\_G↔  
ROUP, [19](#)  
LWT\_REMOTE\_REMOVE\_EVENT\_FROM GRO↔  
UP, [19](#)  
LWT\_REMOTE\_REMOVE\_SENDER\_FROM\_CH↔  
ANNEL, [19](#)  
LWT\_REMOTE\_SIGNAL, [19](#)

## flags

lwt, [13](#)

## id

lwt, [14](#)

## info

lwt, [14](#)

## kthd

lwt, [14](#)

LWT\_INFO\_NCHAN

enums.h, [19](#)

LWT\_INFO\_NRECEIVING

enums.h, [19](#)

LWT\_INFO\_NSENDING

enums.h, [19](#)

LWT\_INFO\_NTHD\_BLOCKED

enums.h, [19](#)

LWT\_INFO\_NTHD\_READY\_POOL

enums.h, [19](#)

LWT\_INFO\_NTHD\_RUNNABLE

enums.h, [19](#)

LWT\_INFO\_NTHD\_ZOMBIES

enums.h, [19](#)

LWT\_INFO\_REAPER\_READY

enums.h, [19](#)

LWT\_JOIN

enums.h, [19](#)

LWT\_NOJOIN

enums.h, [19](#)

LWT\_REMOTE\_ADD\_BLOCKED\_SENDER\_TO\_CHA↔  
NNEL

enums.h, [19](#)

LWT\_REMOTE\_ADD\_CHANNEL\_TO\_GROUP

enums.h, [19](#)

LWT\_REMOTE\_ADD\_EVENT\_TO\_GROUP

enums.h, [19](#)

LWT\_REMOTE\_ADD\_SENDER\_TO\_CHANNEL

enums.h, [19](#)

LWT\_REMOTE\_REMOVE\_BLOCKED\_SENDER\_FRO↔  
M\_CHANNEL

enums.h, [19](#)

LWT\_REMOTE\_REMOVE\_CHANNEL\_FROM\_GROUP

enums.h, [19](#)

LWT\_REMOTE\_REMOVE\_EVENT\_FROM\_GROUP

enums.h, [19](#)

LWT\_REMOTE\_REMOVE\_SENDER\_FROM\_CHANN↔  
EL

enums.h, [19](#)

LWT\_REMOTE\_SIGNAL

enums.h, [19](#)

## lwt

args, [13](#)

flags, [13](#)

id, [14](#)

info, [14](#)

kthd, [14](#)

parent, [14](#)

## main.c

SERVER\_TYPE\_ONE, [51](#)

SERVER\_TYPE\_TWO, [51](#)

parent

lwt, [14](#)

SERVER\_TYPE\_ONE

main.c, [51](#)

SERVER\_TYPE\_TWO

main.c, [51](#)