LWT Threads

1.0

Generated by Doxygen 1.8.8

Fri Apr 3 2015 21:25:39

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 event Struct Reference

Event data.

`#include <lwt_cgrp.h>`

Data Fields

- struct event * previous_event
- struct event * next_event
- lwt_chan_t channel
- void * data

3.1.1 Detailed Description

Event data.

3.1.2 Field Documentation

3.1.2.1 lwt_chan_t event::channel

The channel with the new event

3.1.2.2 void* event::data

The data being added to the channel

3.1.2.3 struct event* event::next_event

The next event

3.1.2.4 struct event* event::previous_event

The previous event

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

- lwt_cgrp.h

3.2 lwt_cgrp_t Struct Reference

Channel group for handling events within a group.

`#include <lwt_cgrp.h>`

Data Fields

- lwt_chan_t channel_head
- lwt_chan_t channel_tail
- struct event * event_head
- struct event * event_tail
- lwt_t waiting_thread

3.2.1 Detailed Description

Channel group for handling events within a group.

3.2.2 Field Documentation

3.2.2.1 lwt_chan_t lwt_cgrp_t::channel_head

Head of the list of channels

3.2.2.2 lwt_chan_t lwt_cgrp_t::channel_tail

Tail of the list of channels

3.2.2.3 struct event* lwt_cgrp_t::event_head

Head of the event queue

3.2.2.4 struct event* lwt_cgrp_t::event_tail

Tail of the event queue

3.2.2.5 lwt_t lwt_cgrp_t::waiting_thread

Waiting thread

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

- lwt_cgrp.h

3.3 lwt_chan_t Struct Reference

The channel for synchronous and asynchronous communication.

`#include <lwt_chan.h>`

Data Fields

- lwt_t senders_head
- lwt_t senders_tail
- int snd_cnt
- lwt_t blocked_senders_head
- lwt_t blocked_senders_tail
- lwt_t receiver
- void * sync_buffer
- void ** async_buffer
- int start_index
- int end_index
- int buffer_size
- int num_entries
- lwt_chan_t previous_sibling
- lwt_chan_t next_sibling
- lwt_cgrp_t channel_group
- lwt_chan_t previous_channel_in_group
- lwt_chan_t next_channel_in_group
- void * mark
- int has_event

3.3.1 Detailed Description

The channel for synchronous and asynchronous communication.

3.3.2 Field Documentation

3.3.2.1 void** lwt_chan_t::async_buffer

Async Buffer to be passed to the channel

3.3.2.2 lwt_t lwt_chan_t::blocked_senders_head

The head of the blocked senders

3.3.2.3 lwt_t lwt_chan_t::blocked_senders_tail

The tail of the blocked senders

3.3.2.4 int lwt_chan_t::buffer_size

Size of the buffer

3.3.2.5 lwt_cgrp_t lwt_chan_t::channel_group

Channel group

3.3.2.6 int lwt_chan_t::end_index

End index of the buffer

3.3.2.7 int lwt_chan_t::has_event

Currently has event

3.3.2.8 void* lwt_chan_t::mark

Mark for channel

3.3.2.9 lwt_chan_t lwt_chan_t::next_channel_in_group

Next channel in group

3.3.2.10 lwt_chan_t lwt_chan_t::next_sibling

Next sibling channel

3.3.2.11 int lwt_chan_t::num_entries

Current number of entries in buffer

3.3.2.12 lwt_chan_t lwt_chan_t::previous_channel_in_group

Previous channel in group

3.3.2.13 lwt_chan_t lwt_chan_t::previous_sibling

Previous sibling channel

3.3.2.14 lwt_t lwt_chan_t::receiver

The receiving thread

3.3.2.15 lwt_t lwt_chan_t::senders_head

The list of senders head

3.3.2.16 lwt_t lwt_chan_t::senders_tail

The list of senders tail

3.3.2.17 int lwt_chan_t::snd_cnt

The number of senders

3.3.2.18 int lwt_chan_t::start_index

Start index of the buffer

3.3.2.19 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:

- lwt_chan.h

3.4 lwt_t Struct Reference

The Lightweight Thread (LWT) struct.

`#include <lwt.h>`

Data Fields

- long * max_addr_thread_stack
- long * min_addr_thread_stack
- long * thread_sp
- lwt_flags_t flags
- lwt_t parent
- lwt_t children
- lwt_t previous_sibling
- lwt_t next_sibling
- lwt_t previous_current
- lwt_t next_current
- lwt_t previous_runnable
- lwt_t next_runnable

- lwt_t previous_ready_pool_thread
- lwt_t next_ready_pool_thread
- lwt_t previous_sender
- lwt_t next_sender
- lwt_t previous_blocked_sender
- lwt_t next_blocked_sender
- lwt_chan_t receiving_channels
- lwt_fnt_t start_routine
- void * args
- void * return_value
- lwt_info_t info
- int id

3.4.1 Detailed Description

The Lightweight Thread (LWT) struct.

3.4.2 Field Documentation

3.4.2.1 void* lwt_t::args

The args for the start_routine

3.4.2.2 lwt_t lwt_t::children

List of children threads

3.4.2.3 lwt_flags_t lwt_t::flags

The flags associated with the lwt

3.4.2.4 int lwt_t::id

The id of the thread

3.4.2.5 lwt_info_t lwt_t::info

The current status of the thread

3.4.2.6 long* lwt_t::max_addr_thread_stack

Pointer to the max address of the stack

3.4.2.7 long* lwt_t::min_addr_thread_stack

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

3.4.2.8 lwt_t lwt_t::next_blocked_sender

Next blocked sender thread

3.4.2.9 lwt_t lwt_t::next_current

Next current thread

3.4.2.10 lwt_t lwt_t::next_ready_pool_thread

Next ready pool thread

3.4.2.11 lwt_t lwt_t::next_runnable

Next runnable thread

3.4.2.12 lwt_t lwt_t::next_sender

Next sender thread

3.4.2.13 lwt_t lwt_t::next_sibling

Next sibling

3.4.2.14 lwt_t lwt_t::parent

Parent thread

3.4.2.15 lwt_t lwt_t::previous_blocked_sender

Previous blocked sender thread

3.4.2.16 lwt_t lwt_t::previous_current

Previous current thread

3.4.2.17 lwt_t lwt_t::previous_ready_pool_thread

Previous ready pool thread

3.4.2.18 lwt_t lwt_t::previous_runnable

Previous runnable thread

3.4.2.19 lwt_t lwt_t::previous_sender

Previous sender thread

3.4.2.20 lwt_t lwt_t::previous_sibling

Previous sibling

3.4.2.21 lwt_chan_t lwt_t::receiving_channels

List of receiving channels associated with the thread

3.4.2.22 void* lwt_t::return_value

The return value from the routine

3.4.2.23 lwt_fnt_t lwt_t::start_routine

The start routine for the thread to run

3.4.2.24 long* lwt_t::thread_sp

The current thread stack pointer for the thread

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

- lwt.h

3.5 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.

3.5.1 Detailed Description

Struct for passing the args to merge sort around.

3.5.2 Field Documentation

3.5.2.1 int msort_args::begin_index

The begin index of the segment.

3.5.2.2 int* msort_args::data

The int array holding randomly generated data.

3.5.2.3 int msort_args::end_index

THe end index of the segment.

3.5.2.4 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 lwt.c File Reference

```
#include "lwt.h"
#include "lwt_chan.h"
#include "lwt_cgrp.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.
- 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.
- 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.

4.1.1 Macro Definition Documentation

4.1.1.1 #define DEFAULT_ID -1

The default id provided to threads before actually generating them.

4.1.1.2 #define INIT_ID 1

The initial thread id.

4.1.1.3 #define POOL_SIZE 100

The size of the pool.

4.1.2 Function Documentation

4.1.2.1 __attribute__ ((constructor))

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

4.1.2.2 __attribute__ ((destructor))

Cleans up all remaining threads on exit.

4.1.2.3 void __cleanup_joined_thread (lwt_t *lwt*)

Cleans up the thread on join.

Parameters

lwt	The thread to join on

4.1.2.4 void __init_lwt_main (lwt_t *thread*)

Initializes the main thread.

Parameters

thread	The main thread

4.1.2.5 void __init_new_lwt (lwt_t *thread*)

Initializes the provided thread.

Parameters

thread	The thread to init

4.1.2.6 void __insert_runnable_tail (lwt_t *thread*)

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

Parameters

thread	The new thread to be inserted in the list of runnable threads

4.1.2.7 void __lwt_dispatch (lwt_t *next,* lwt_t *current*)

Dispatch function for switching between threads.

Parameters

next	The next thread to switch to
current	The current thread

4.1.2.8 void __lwt_schedule (void)

Schedules the next_current thread to switch to and dispatches.

4.1.2.9 void * __lwt_stack_get (void)

Allocates the stack for a LWT and returns it.

4.1.2.10 void __lwt_stack_return (void * *stack*)

Frees the provided stack.

Parameters

stack	The LWT stack to free

4.1.2.11 void __lwt_trampoline (void)

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

4.1.2.12 void __reinit_lwt (lwt_t *thread*)

Reinitializes the given thread.

Parameters

thread	The thread to reinitialize

4.1.2.13 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

fn	The function pointer to use
data	The data to the function
flags	The flags to be associated with the thread

Returns

A pointer to the initialized LWT

4.1.2.14 lwt_t lwt_current ( )

Gets the current thread.

Returns

The current thread

4.1.2.15 void lwt_die (void * *value*)

Prepares the current thread to be cleaned up.

4.1.2.16 int lwt_id (lwt_t *thread*)

Gets the thread id.

Returns

The id of the thread

4.1.2.17 int lwt_info (lwt_info_t *t*)

Gets the counts of the info.

Parameters

t	The info enum to get the counts

Returns

The count for the info enum provided

See also

lwt_info_t

4.1.2.18 void* lwt_join (lwt_t *thread*)

Joins the provided thread.

Parameters

thread	The thread to join on

4.1.2.19 int lwt_yield (lwt_t *lwt*)

Yields to the provided LWT.

Parameters

lwt	The thread to yield to

Note

Will just schedule normally if LWT_NULL is provided

Returns

0 if successful

4.2 lwt.h File Reference

```
#include "stdlib.h"
```

Data Structures

- struct lwt_t

  The Lightweight Thread (LWT) struct.

Macros

- #define PAGE_SIZE 4096
- #define NUM_PAGES 5
- #define STACK_SIZE PAGE_SIZE∗NUM_PAGES
- #define LWT_NULL NULL

Typedefs

- typedef void ∗(∗ lwt_fnt_t)(void ∗)

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 }

  The various statuses for a LWT.
- enum lwt_flags_t { LWT_JOIN = 0, LWT_NOJOIN = 1 }

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 __insert_runnable_tail (lwt_t)

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

4.2.1 Macro Definition Documentation

4.2.1.1 #define LWT_NULL NULL

Null id for yields

4.2.1.2 #define NUM_PAGES 5

Number of pages to allocate to the stack

4.2.1.3 #define PAGE_SIZE 4096

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

4.2.1.4 #define STACK_SIZE PAGE_SIZE∗NUM_PAGES

Size of the stack

4.2.2 Typedef Documentation

4.2.2.1 typedef void∗(∗ lwt_fnt_t)(void ∗)

4.2.3 Enumeration Type Documentation

4.2.3.1 enum lwt_flags_t

flags for determining if the lwt is joinable

Enumerator

LWT_JOIN lwt is joinable

LWT_NOJOIN lwt is not joinable

4.2.3.2 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

4.2.4 Function Documentation

4.2.4.1 void __insert_runnable_tail (lwt_t *thread*)

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

Parameters

thread	The new thread to be inserted in the list of runnable threads

4.2.4.2 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

fn	The function pointer to use
data	The data to the function
flags	The flags to be associated with the thread

Returns

A pointer to the initialized LWT

4.2.4.3 lwt_t lwt_current ( )

Gets the current thread.

Returns

The current thread

4.2.4.4 void lwt_die (void *)

Prepares the current thread to be cleaned up.

4.2.4.5 int lwt_id (lwt_t *thread*)

Gets the thread id.

Returns

The id of the thread

4.2.4.6 int lwt_info (lwt_info_t *t*)

Gets the counts of the info.

Parameters

t	The info enum to get the counts

Returns

The count for the info enum provided

See also

lwt_info_t

4.2.4.7 void* lwt_join (lwt_t *thread*)

Joins the provided thread.

Parameters

thread	The thread to join on

4.2.4.8 int lwt_yield (lwt_t *lwt*)

Yields to the provided LWT.

Parameters

lwt	The thread to yield to

Note

Will just schedule normally if LWT_NULL is provided

Returns

0 if successful

4.3 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"
```

Functions

- void __init_event (lwt_chan_t channel, void *data)

  Initializes the event for when data is added to the channel.
- void free_event (struct event *event)

  Removes the event from the group.
- void __pop_event (lwt_cgrp_t group)

  Pops the 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.

4.3.1 Function Documentation

4.3.1.1 void __init_event (lwt_chan_t *channel*, void * *data*)

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

Parameters

channel	The channel with the new data
data	The data being inserted

4.3.1.2 void __pop_event (lwt_cgrp_t *group*)

Pops the event from the group.

Parameters

group	The channel group to alter

4.3.1.3 void free_event (struct event * *event*)

Removes the event from the group.

Parameters

event	The event being removed

4.3.1.4 lwt_cgrp_t lwt_cgrp ( )

Creates a group of channels.

Returns

The group of channels

Note

By default, the group is empty

4.3.1.5 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

group	The group to add the channel to
channel	The channel to add

Returns

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

4.3.1.6 int lwt_cgrp_free (lwt_cgrp_t *group*)

Frees the group if there are no pending events.

Parameters

group	The channel group to free

Returns

0 if successful; -1 if there are pending events

4.3.1.7 int lwt_cgrp_rem (lwt_cgrp_t *group,* lwt_chan_t *channel*)

Removes the channel from the group.

Parameters

group	The group to remove the channel from
channel	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

4.3.1.8 lwt_chan_t lwt_cgrp_wait (lwt_cgrp_t *group*)

Waits until there is a pending event in the queue.

Parameters

group	The group to wait for

Returns

The event in the queue

4.3.1.9 void* lwt_chan_mark_get (lwt_chan_t *channel*)

Grabs the mark from the channel.

Parameters

channel	The channel to read

4.3.1.10 void lwt_chan_mark_set (lwt_chan_t *channel*, void * *mark*)

Marks the channel.

Parameters

channel	The channel to mark
mark	The marker to set

4.4 lwt_cgrp.h File Reference

```
#include "lwt_chan.h"
```

Data Structures

- struct event

  Event data.
- struct lwt_cgrp_t

  Channel group for handling events within a group.

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, void *)

  Initializes the event for when data is added to the channel.
- void __pop_event (lwt_cgrp_t)

  Pops the event from the group.

4.4.1 Function Documentation

4.4.1.1 void __init_event (lwt_chan_t *channel,* void * *data*)

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

Parameters

channel	The channel with the new data
data	The data being inserted

4.4.1.2 void __pop_event (lwt_cgrp_t *group*)

Pops the event from the group.

Parameters

group	The channel group to alter

4.4.1.3 lwt_cgrp_t lwt_cgrp ( )

Creates a group of channels.

Returns

The group of channels

Note

By default, the group is empty

4.4.1.4 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

group	The group to add the channel to
channel	The channel to add

Returns

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

4.4.1.5 int lwt_cgrp_free (lwt_cgrp_t *group*)

Frees the group if there are no pending events.

Parameters

group	The channel group to free

Returns

0 if successful; -1 if there are pending events

4.4.1.6 int lwt_cgrp_rem (lwt_cgrp_t *group,* lwt_chan_t *channel*)

Removes the channel from the group.

Parameters

group	The group to remove the channel from
channel	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

4.4.1.7 lwt_chan_t lwt_cgrp_wait (lwt_cgrp_t *group*)

Waits until there is a pending event in the queue.

Parameters

group	The group to wait for

Returns

The event in the queue

4.4.1.8 void* lwt_chan_mark_get (lwt_chan_t *channel*)

Grabs the mark from the channel.

Parameters

channel	The channel to read

4.4.1.9 void lwt_chan_mark_set (lwt_chan_t *channel,* void * *mark*)

Marks the channel.

Parameters

channel	The channel to mark
mark	The marker to set

4.5 lwt_chan.c File Reference

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

Functions

- void __remove_channel (lwt_chan_t channel)

  Remove the channel from the list of channels.
- void __remove_from_blocked_sender (lwt_chan_t c, lwt_t thread)

  Removes the thread from the list of blocked sender threads.
- 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.

4.5.1 Function Documentation

4.5.1.1 void* __pop_data_from_async_buffer (lwt_chan_t *c*)

Pops the data into the buffer.

Parameters

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

4.5.1.2 void __remove_channel (lwt_chan_t *channel*)

Remove the channel from the list of channels.

Parameters

channel	The channel to be removed

4.5.1.3 void __remove_from_blocked_sender (lwt_chan_t *c*, lwt_t *thread*)

Removes the thread from the list of blocked sender threads.

Parameters

channel	The channel from which the sender will be removed
thread	The thread to be removed

4.5.1.4 lwt_chan_t lwt_chan (int *sz*)

Creates the channel on the receiving thread.

Parameters

sz	The size of the buffer

Returns

A pointer to the initialized channel

4.5.1.5 void lwt_chan_deref (lwt_chan_t *c*)

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

Parameters

c	The channel to deallocate

4.5.1.6 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

fn	The function to use to create the thread
c	The channel to send
flags	The flags for the thread

Returns

The thread to return

4.5.1.7 void* lwt_rcv (lwt_chan_t *c*)

Receives the data from the channel and returns it.

Parameters

c	The channel to receive from

Returns

The data from the channel

4.5.1.8 lwt_chan_t lwt_rcv_chan (lwt_chan_t *c*)

Receives the data over the channel.

Parameters

c	The channel to use for receiving

Returns

The channel being sent over c

4.5.1.9 int lwt_snd (lwt_chan_t *c*, void * *data*)

Sends the data over the channel to the receiver.

Parameters

c	The channel to use for sending
data	The data for sending

Returns

-1 if there is no receiver; 0 if successful

4.5.1.10 int lwt_snd_chan (lwt_chan_t *c*, lwt_chan_t *sending*)

Sends sending over the channel c.

Parameters

c	The channel to send sending across
sending	The channel to send

4.6 lwt_chan.h File Reference

```
#include "lwt.h"
```

Data Structures

- struct lwt_chan_t

  The channel for synchronous and asynchronous communication.

Typedefs

- typedef void *(* lwt_chan_fn_t)(lwt_chan_t)

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 __remove_channel (lwt_chan_t)

  Remove the channel from the list of channels.
- void * __pop_data_from_async_buffer (lwt_chan_t)

  Pops the data into the buffer.
- void __remove_from_blocked_sendera (lwt_chan_t, lwt_t)

4.6.1 Typedef Documentation

4.6.1.1 typedef void*(* lwt_chan_fn_t)(lwt_chan_t)

4.6.2 Function Documentation

4.6.2.1 void* __pop_data_from_async_buffer (lwt_chan_t *c*)

Pops the data into the buffer.

Parameters

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

4.6.2.2 void __remove_channel (lwt_chan_t *channel*)

Remove the channel from the list of channels.

Parameters

channel	The channel to be removed

4.6.2.3 void __remove_from_blocked_sendera (lwt_chan_t , lwt_t)

4.6.2.4 lwt_chan_t lwt_chan (int *sz*)

Creates the channel on the receiving thread.

Parameters

sz	The size of the buffer

Returns

A pointer to the initialized channel

4.6.2.5 void lwt_chan_deref (lwt_chan_t *c*)

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

Parameters

c	The channel to deallocate

4.6.2.6 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

fn	The function to use to create the thread
c	The channel to send
flags	The flags for the thread

Returns

The thread to return

4.6.2.7 void* lwt_rcv (lwt_chan_t *c*)

Receives the data from the channel and returns it.

Parameters

c	The channel to receive from

Returns

The data from the channel

4.6.2.8 lwt_chan_t lwt_rcv_chan (lwt_chan_t *c*)

Receives the data over the channel.

Parameters

c	The channel to use for receiving

Returns

The channel being sent over c

4.6.2.9 int lwt_snd (lwt_chan_t *c*, void * *data*)

Sends the data over the channel to the receiver.

Parameters

c	The channel to use for sending
data	The data for sending

Returns

-1 if there is no receiver; 0 if successful

4.6.2.10 int lwt_snd_chan (lwt_chan_t *c*, lwt_chan_t *sending*)

Sends sending over the channel c.

Parameters

c	The channel to send sending across
sending	The channel to send

4.7 main.c File Reference

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

Macros

- #define rdtscll(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

4.7.1 Macro Definition Documentation

4.7.1.1 #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)
```

4.7.1.2 #define ITER 10000

4.7.1.3 #define rdtscll(*val*) __asm__ __volatile__("rdtsc" : "=A" (val))

4.7.2 Function Documentation

4.7.2.1 void* fn_bounce (void * *d*)

4.7.2.2 void* fn_identity (void * *d*)

4.7.2.3 void* fn_join (void * *d*)

4.7.2.4 void* fn_nested_joins (void * *d*)

4.7.2.5 void* fn_null (void * *d*)

4.7.2.6 void* fn_sequence (void * *d*)

4.7.2.7 int main (void)

4.7.2.8 void test_crt_join_sched (void)

4.7.2.9 void test_perf (void)

4.7.3 Variable Documentation

4.7.3.1 volatile int curr = 0

4.7.3.2 volatile int sched[2] = {0, 0}

4.8 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 rdtscll(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

4.8.1 Macro Definition Documentation

4.8.1.1 #define GRPSZ 3

4.8.1.2 #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)
```

4.8.1.3 #define ITER 10000

4.8.1.4 #define rdtscll(*val*) __asm__ __volatile__("rdtsc" : "=A" (val))

4.8.2 Function Documentation

4.8.2.1 void* fn_async_steam (lwt_chan_t *to*)

4.8.2.2 void* fn_bounce (void * *d*)

4.8.2.3 void* fn_chan (lwt_chan_t *to*)

4.8.2.4 void* fn_grpwait (lwt_chan_t *c*)

4.8.2.5 void* fn_identity (void * *d*)

4.8.2.6 void* fn_join (void * *d*)

4.8.2.7 void* fn_nested_joins (void * *d*)

4.8.2.8 void* fn_null (void * *d*)

4.8.2.9 void* fn_sequence (void * *d*)

4.8.2.10 void* fn_snder (lwt_chan_t *c,* int *v*)

4.8.2.11 void* fn_snder_1 (lwt_chan_t *c*)

4.8.2.12 void* fn_snder_2 (lwt_chan_t *c*)

4.8.2.13 int main (void)

4.8.2.14 void test_crt_join_sched (void)

4.8.2.15 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 allows us to iterate on a channel while there is more data pending.

4.8.2.16 void test_multisend (int *chsz*)

4.8.2.17 void test_perf (void)

4.8.2.18 void test_perf_async_steam (int *chsz*)

4.8.2.19 void test_perf_channels (int *chsz*)

4.8.3 Variable Documentation

4.8.3.1 volatile int curr = 0

4.8.3.2 volatile int sched[2] = {0, 0}

4.9 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 ()

4.9.1 Macro Definition Documentation

4.9.1.1 #define ITER 80

4.9.1.2 #define MERGE_SZ 80

4.9.2 Function Documentation

4.9.2.1 void* child_multiple_channels (lwt_chan_t *main_channel*)

4.9.2.2 void* child_ping (lwt_chan_t *main_channel*)

Ping channel test.

Parameters

main_channel	The channel from the main thread

Returns

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

4.9.2.3 void* child_pong (lwt_chan_t *main_channel*)

Receives a count, updates it and sends it back.

Parameters

main_channel	The channel from the main thread

Returns

0 if successful

4.9.2.4 int main (void)

Main function

4.9.2.5 void merge_sort_test ( )

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

4.9.2.6 void* msort (lwt_chan_t *main_channel*)

Merge sort in parallel.

Parameters

main_channel	The channel from the main thread

Returns

0 if successful

Note

Adapted from wikipedia: http://en.wikipedia.org/wiki/Merge_sort#Parallel_merge_sort)

4.9.2.7 void multiple_channels_test ( )

4.9.2.8 void multiple_channels_test_v2 ( )

4.9.2.9 void multiple_channels_test_v3 ( )

4.9.2.10 void ping_pong_test ( )

Runs the ping/pong test.

Index