Word Counter

Generated by Doxygen 1.9.6

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 hashmap_element_s Struct Reference

`#include <hm.h>`

Public Attributes

- char * key
- void * data

3.1.1 Member Data Documentation

3.1.1.1 data

`void* hashmap_element_s::data`

3.1.1.2 key

`char* hashmap_element_s::key`

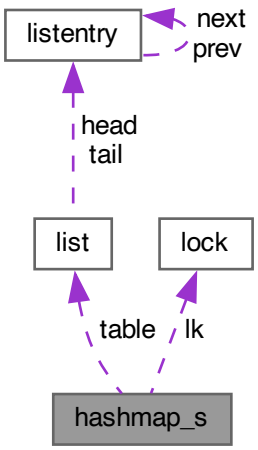
The documentation for this struct was generated from the following file:

- include/hm.h

3.2 hashmap_s Struct Reference

`#include <hm.h>`

Collaboration diagram for hashmap_s:



Public Attributes

- struct list * table [SZ]
- struct lock * lk [SZ]

3.2.1 Member Data Documentation

3.2.1.1 lk

`struct lock* hashmap_s::lk[SZ]`

3.2.1.2 table

`struct list* hashmap_s::table[SZ]`

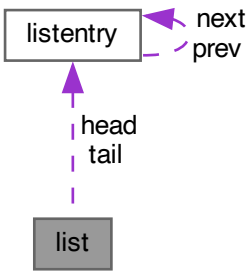
The documentation for this struct was generated from the following file:

- include/hm.h

3.3 list Struct Reference

`#include <list.h>`

Collaboration diagram for list:



Public Attributes

- struct listentry * head

  head of the list
- struct listentry * tail

  tail of the list

3.3.1 Member Data Documentation

3.3.1.1 head

`struct listentry* list::head`

head of the list

3.3.1.2 tail

`struct listentry* list::tail`

tail of the list

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

- include/list.h

3.4 listentry Struct Reference

`#include <list.h>`

Collaboration diagram for listentry:

Public Attributes

- void * data
  data for this entry
- struct listentry * prev
  previous entry
- struct listentry * next
  next entry

3.4.1 Member Data Documentation

3.4.1.1 data

`void* listentry::data`

data for this entry

3.4.1.2 next

`struct listentry* listentry::next`

next entry

3.4.1.3 prev

```
struct listentry* listentry::prev
```

previous entry

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

- include/list.h

3.5 lock Struct Reference

```
#include <mythread.h>
```

Public Attributes

- void * c

3.5.1 Member Data Documentation

3.5.1.1 c

```
void* lock::c
```

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

- include/mythread.h

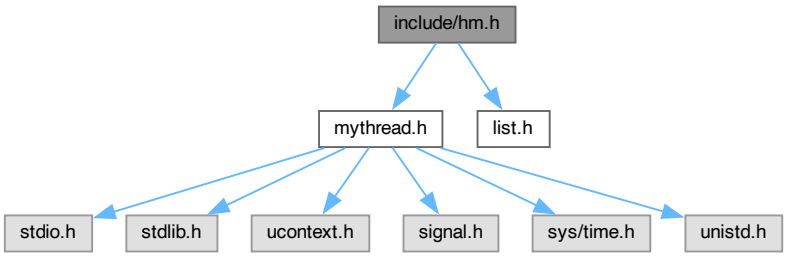
Chapter 4

File Documentation

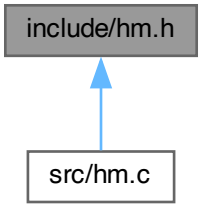
4.1 include/hm.h File Reference

```
#include "mythread.h"
#include "list.h"
```

Include dependency graph for hm.h:



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



Classes

- struct hashmap_element_s
- struct hashmap_s

Macros

- #define SZ 4096

Functions

- int hashmap_create (struct hashmap_s *const out_hashmap)

  Creates a empty hashmap.
- int hashmap_put (struct hashmap_s *const hashmap, const char *key, void *data)

  puts entry/update entry in hashmap
- void * hashmap_get (struct hashmap_s const *hashmap, const char *key)

  obtains the data (no of times a word repeats) from the hashmap
- void hashmap_iterator (struct hashmap_s *const hashmap, int(*f)(struct hashmap_element_s *const))

  iterates through the hashmap and calls the function f on each element
- int acquire_bucket (struct hashmap_s *const hashmap, const char *key)

  acquires the lock of the bucket corresponding to the key Calls lock_acquire on the lock corresponding to the bucket
- int release_bucket (struct hashmap_s *const hashmap, const char *key)

  releases the lock of the bucket corresponding to the key Calls lock_release on the lock corresponding to the bucket

4.1.1 Macro Definition Documentation

4.1.1.1 SZ

```
#define SZ 4096
```

4.1.2 Function Documentation

4.1.2.1 acquire_bucket()

```
int acquire_bucket (
            struct hashmap_s *const hashmap,
            const char * key )
```

acquires the lock of the bucket corresponding to the key Calls lock_acquire on the lock corresponding to the bucket

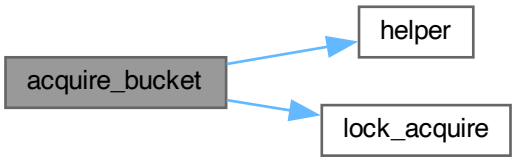
Parameters

in	*hashmap*	struct hashmap_s *const hashmap (hashmap whose lock is to be acquired)
in	*key*	const char* key (key of the entry whose lock is to be acquired)

Returns

* int

Here is the call graph for this function:



4.1.2.2 hashmap_create()

```
int hashmap_create (
            struct hashmap_s *const out_hashmap )
```

Creates a empty hashmap.

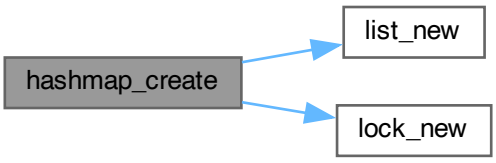
Parameters

in	*out_hashmap*	struct hashmap_s *const element creating a hashmap with 4096 buckets and initializing the locks and linked lists

Returns

0

Here is the call graph for this function:



4.1.2.3 hashmap_get()

```
void * hashmap_get (
            struct hashmap_s const * hashmap,
            const char * key )
```

obtains the data (no of times a word repeats) from the hashmap

Parameters

hashmap	struct hashmap_s const* hashmap(hashmap from which the data is to be obtained)
key	const char* key (key of the entry whose data is to be obtained) uses while loopn to iterate through the linked list and returns the data if found else returns NULL

Returns

void*

Here is the call graph for this function:

4.1.2.4 hashmap_iterator()

```
void hashmap_iterator (
            struct hashmap_s *const hashmap,
            int(*)(struct hashmap_element_s *const) f )
```

iterates through the hashmap and calls the function f on each element

Parameters

in	*hashmap*	struct hashmap_s* const hashmap (hashmap whose entries are to be iterated)
in	*f*	int (*f)(struct hashmap_element_s *const (function pointer to the function to be called on each element) We choose f as pointer to Readfile function which is called on each element of the hashmap Iterate through the linked list corresponding to each bucket and call f on each element

Returns

void

4.1.2.5 hashmap_put()

```
int hashmap_put (
            struct hashmap_s *const hashmap,
            const char * key,
            void * data )
```

puts entry/update entry in hashmap

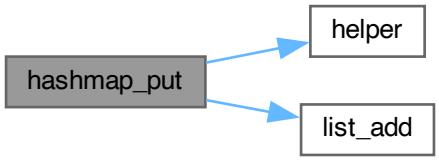
Parameters

in	*hashmap*	struct hashmap_s *const hashmap (hashmap whose entries are to be updated)
in	*key*	const char* key (key of the entry to be added/increased)
in	*data*	void* data (data of the entry to be added/increased) Find searches the key in the corresponding bucket and if found updates the data else creates a new entry

Returns

int 0

Here is the call graph for this function:



4.1.2.6 release_bucket()

```
int release_bucket (
            struct hashmap_s *const hashmap,
            const char * key )
```

releases the lock of the bucket corresponding to the key Calls lock_release on the lock corresponding to the bucket

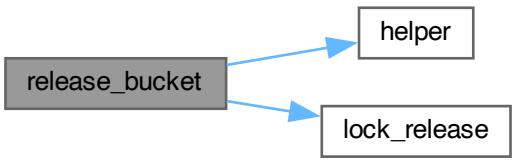
Parameters

hashmap[in]	struct hashmap_s *const hashmap (hashmap whose lock is to be released)
key[in]	const char* key (key of the entry whose lock is to be released)

Returns

* int

Here is the call graph for this function:



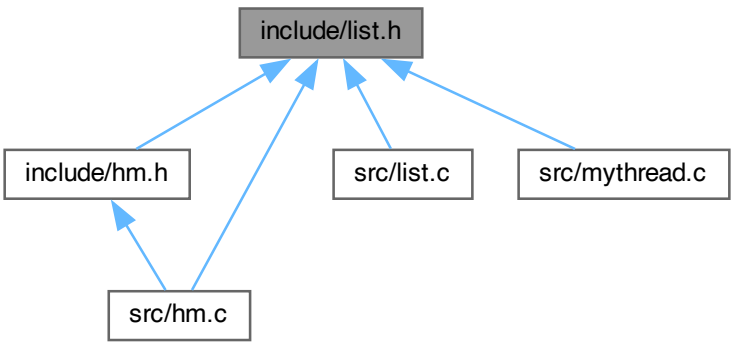
4.2 hm.h

Go to the documentation of this file.

```
00001 #include "mythread.h"
00002 #include "list.h"
00003 #define SZ 4096
00004
00005 struct hashmap_element_s {
00006   char *key;
00007   void *data;
00008 };
00009
00010 struct hashmap_s {
00011   struct list* table[SZ];
00012   struct lock* lk[SZ];
00013 };
00014
00015
00016 int hashmap_create(struct hashmap_s *const out_hashmap);
00017 int hashmap_put(struct hashmap_s *const hashmap, const char* key, void* data);
00018 void* hashmap_get(struct hashmap_s const* hashmap, const char* key);    // Fetch value of a key from
      hashmap
00019 void hashmap_iterator(struct hashmap_s* const hashmap,
00020                       int (*f)(struct hashmap_element_s *const));
00021
00022 int acquire_bucket(struct hashmap_s *const hashmap, const char* key);
00023 int release_bucket(struct hashmap_s *const hashmap, const char* key);
```

4.3 include/list.h File Reference

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



Classes

- struct list
- struct listentry

Functions

- void list_rm (struct list *l, struct listentry *e)

  Linked list implementation Linked list implementation for the part 3 of the project.
- struct listentry * list_add (struct list *l, void *data)

  Adds an element to the list.
- struct list * list_new ()

  Creates a empty list.
- int is_empty (struct list *l)

  Checks if the list is empty.

4.3.1 Function Documentation

4.3.1.1 is_empty()

```
int is_empty (
            struct list * l )
```

Checks if the list is empty.

Parameters

in	*l*	struct list* l -> the linked list to be checked

Returns

int 1/true if empty else 0/false

4.3.1.2 list_add()

```
struct listentry * list_add (
            struct list * l,
            void * data )
```

Adds an element to the list.

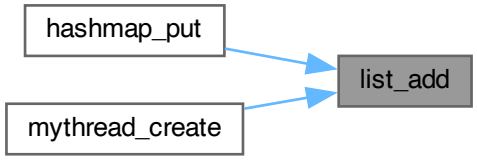
Parameters

in	*l*	struct list* l -> the linked list to which the element is to be added
in	*data*	void* data -> the data to be added to the list

Returns

struct listentry* new_entry -> the new entry added to the list

Here is the caller graph for this function:



4.3.1.3 list_new()

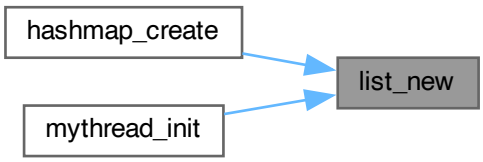
```
struct list * list_new ( )
```

Creates a empty list.

Returns

struct list* l -> the new list created

Here is the caller graph for this function:



4.3.1.4 list_rm()

```
void list_rm (
            struct list * l,
            struct listentry * e )
```

Linked list implementation Linked list implementation for the part 3 of the project.

Removes an element from the list Set the attributes of the previous and next element of the element to be removed

Parameters

in	*l*	struct list* l -> the linked list whose element is to be removed
in	*e*	struct listentry* e -> the element to be removed

Returns

void

4.4 list.h

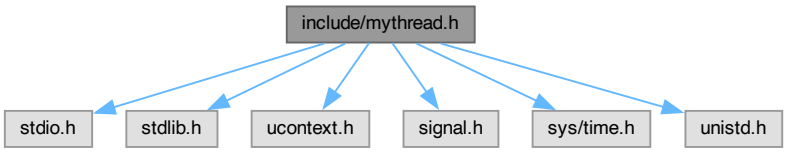
Go to the documentation of this file.

```
00001 #ifndef LIST_H
00002 #define LIST_H
00003 struct list {
00004     struct listentry *head;
00005     struct listentry *tail;
00006 };
00007
00008 struct listentry {
00009     void *data;
00010     struct listentry *prev;
00011     struct listentry *next;
00012 };
00013
00014 void list_rm(struct list *l, struct listentry *e);     // Remove an item from the list
00015 struct listentry *list_add(struct list *l, void *data);   // Add an item to the list
00016 struct list *list_new();                    // Return an initialized list
00017 int is_empty(struct list *l);                  // Check if list is empty or not
00018 #endif
```

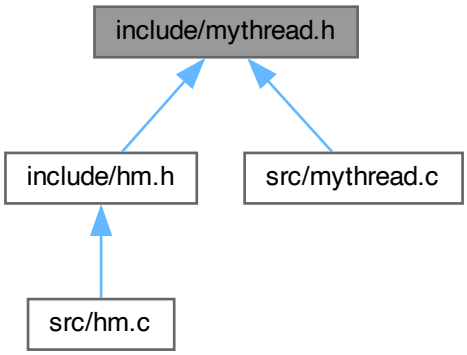
4.5 include/mythread.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <ucontext.h>
#include <signal.h>
#include <sys/time.h>
#include <unistd.h>
```

Include dependency graph for mythread.h:



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



Classes

- struct lock

Functions

- void mythread_init ()
- void * mythread_create (void func(void *), void *arg)
- void mythread_join ()
- void mythread_yield ()
- struct lock * lock_new ()
- void lock_acquire (struct lock *lk)
- int lock_release (struct lock *lk)

4.5.1 Function Documentation

4.5.1.1 lock_acquire()

```
void lock_acquire (
            struct lock * lk )
```

Here is the caller graph for this function:

4.5.1.2 lock_new()

```
struct lock * lock_new ( )
```

Here is the caller graph for this function:

4.5.1.3 lock_release()

```
int lock_release (
            struct lock * lk )
```

Here is the caller graph for this function:

4.5.1.4 mythread_create()

```
void * mythread_create (
            void  funcvoid *,
            void * arg )
```

Here is the call graph for this function:

4.5.1.5 mythread_init()

```
void mythread_init ( )
```

Here is the call graph for this function:

4.5.1.6 mythread_join()

```
void mythread_join ( )
```

4.5.1.7 mythread_yield()

```
void mythread_yield ( )
```

4.6 mythread.h

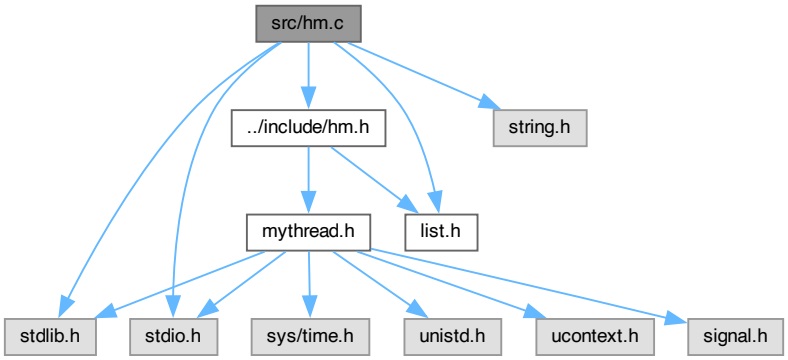
Go to the documentation of this file.

```
00001 #ifndef THREAD_H
00002 #define THREAD_H
00003
00004 #include<stdio.h>
00005 #include<stdlib.h>
00006 #include<ucontext.h>
00007 #include<signal.h>
00008 #include<sys/time.h>
00009 #include<unistd.h>
00010
00011
00012
00013 void mythread_init();
00014 void* mythread_create(void func(void*), void* arg);
00015 void mythread_join();
00016 void mythread_yield();
00017
00018 struct lock {
00019     void* c;
00020 };
00021 struct lock* lock_new();
00022 void lock_acquire(struct lock* lk);
00023 int lock_release(struct lock* lk);
00024
00025 #endif
```

4.7 src/hm.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "../include/hm.h"
#include "../include/list.h"
#include <string.h>
```

Include dependency graph for hm.c:



Macros

- #define SZ 4096

  Hashmap implementation Hashmap implementation for the part 3 of the project.

Functions

- int helper (const char ∗str1)

  hashing function hashing function to get the bucket index of the key
- int hashmap_create (struct hashmap_s ∗const out_hashmap)

  Creates a empty hashmap.
- int hashmap_put (struct hashmap_s ∗const hashmap, const char ∗key, void ∗data)

  puts entry/update entry in hashmap
- void ∗ hashmap_get (struct hashmap_s const ∗hashmap, const char ∗key)

  obtains the data (no of times a word repeats) from the hashmap
- void hashmap_iterator (struct hashmap_s ∗const hashmap, int(∗f)(struct hashmap_element_s ∗const))

  iterates through the hashmap and calls the function f on each element
- int acquire_bucket (struct hashmap_s ∗const hashmap, const char ∗key)

  acquires the lock of the bucket corresponding to the key Calls lock_acquire on the lock corresponding to the bucket
- int release_bucket (struct hashmap_s ∗const hashmap, const char ∗key)

  releases the lock of the bucket corresponding to the key Calls lock_release on the lock corresponding to the bucket

4.7.1 Macro Definition Documentation

4.7.1.1 SZ

```
#define SZ 4096
```

Hashmap implementation Hashmap implementation for the part 3 of the project.

4.7.2 Function Documentation

4.7.2.1 acquire_bucket()

```
int acquire_bucket (
            struct hashmap_s *const hashmap,
            const char * key )
```

acquires the lock of the bucket corresponding to the key Calls lock_acquire on the lock corresponding to the bucket

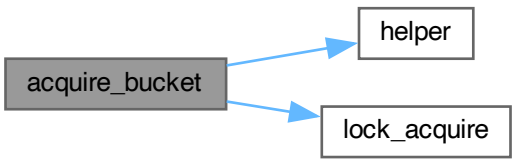
Parameters

in	*hashmap*	struct hashmap_s *const hashmap (hashmap whose lock is to be acquired)
in	*key*	const char* key (key of the entry whose lock is to be acquired)

Returns

* int

Here is the call graph for this function:



4.7.2.2 hashmap_create()

```
int hashmap_create (
            struct hashmap_s *const out_hashmap )
```

Creates a empty hashmap.

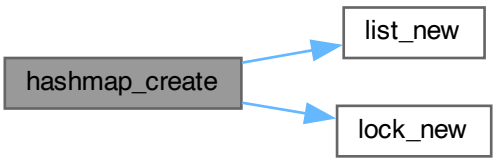
Parameters

in	*out_hashmap*	struct hashmap_s ∗const element creating a hashmap with 4096 buckets and initializing the locks and linked lists

Returns

0

Here is the call graph for this function:



4.7.2.3 hashmap_get()

```
void * hashmap_get (
            struct hashmap_s const * hashmap,
            const char * key )
```

obtains the data (no of times a word repeats) from the hashmap

Parameters

hashmap	struct hashmap_s const∗ hashmap(hashmap from which the data is to be obtained)
key	const char∗ key (key of the entry whose data is to be obtained) uses while loopn to iterate through the linked list and returns the data if found else returns NULL

Returns

void*

Here is the call graph for this function:

4.7.2.4 hashmap_iterator()

```
void hashmap_iterator (
            struct hashmap_s *const hashmap,
            int(*)(struct hashmap_element_s *const) f )
```

iterates through the hashmap and calls the function f on each element

Parameters

in	*hashmap*	struct hashmap_s* const hashmap (hashmap whose entries are to be iterated)
in	*f*	int (*f)(struct hashmap_element_s *const (function pointer to the function to be called on each element) We choose f as pointer to Readfile function which is called on each element of the hashmap Iterate through the linked list corresponding to each bucket and call f on each element

Returns

void

4.7.2.5 hashmap_put()

```
int hashmap_put (
            struct hashmap_s *const hashmap,
            const char * key,
            void * data )
```

puts entry/update entry in hashmap

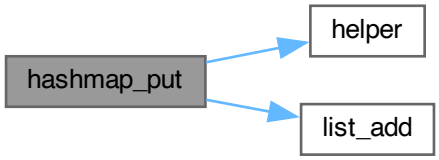
Parameters

in	*hashmap*	struct hashmap_s ∗const hashmap (hashmap whose entries are to be updated)
in	*key*	const char∗ key (key of the entry to be added/increased)
in	*data*	void∗ data (data of the entry to be added/increased) Find searches the key in the corresponding bucket and if found updates the data else creates a new entry

Returns

int 0

Here is the call graph for this function:



4.7.2.6 helper()

```
int helper (
            const char * str1 )
```

hashing function hashing function to get the bucket index of the key

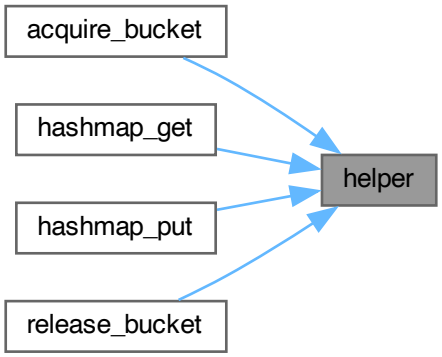
Parameters

in	*str1*	the key const char∗ (The string whose hash value is to be calculated)

Returns

ans the index of the key in the hashmap

Here is the caller graph for this function:



4.7.2.7 release_bucket()

```
int release_bucket (
            struct hashmap_s *const hashmap,
            const char * key )
```

releases the lock of the bucket corresponding to the key Calls lock_release on the lock corresponding to the bucket

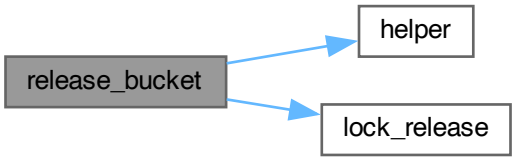
Parameters

hashmap[in]	struct hashmap_s ∗const hashmap (hashmap whose lock is to be released)
key[in]	const char∗ key (key of the entry whose lock is to be released)

Returns

∗ int

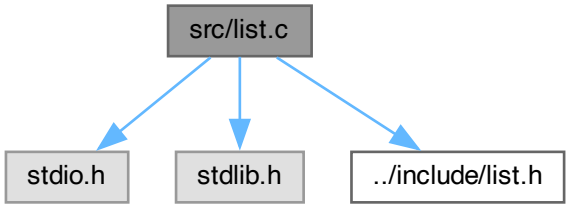
Here is the call graph for this function:



4.8 src/list.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "../include/list.h"
```

Include dependency graph for list.c:



Functions

- void list_rm (struct list ∗l, struct listentry ∗e)

  Linked list implementation Linked list implementation for the part 3 of the project.
- struct listentry ∗ list_add (struct list ∗l, void ∗data)

  Adds an element to the list.
- struct list ∗ list_new ()

  Creates a empty list.
- int is_empty (struct list ∗l)

  Checks if the list is empty.

4.8.1 Function Documentation

4.8.1.1 is_empty()

```
int is_empty (
            struct list * l )
```

Checks if the list is empty.

Parameters

in	*l*	struct list* l -> the linked list to be checked

Returns

int 1/true if empty else 0/false

4.8.1.2 list_add()

```
struct listentry * list_add (
            struct list * l,
            void * data )
```

Adds an element to the list.

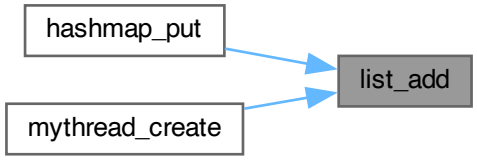
Parameters

in	*l*	struct list* l -> the linked list to which the element is to be added
in	*data*	void* data -> the data to be added to the list

Returns

struct listentry* new_entry -> the new entry added to the list

Here is the caller graph for this function:



4.8.1.3 list_new()

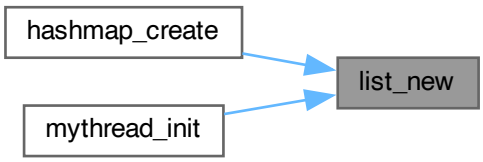
```
struct list * list_new ( )
```

Creates a empty list.

Returns

struct list* l -> the new list created

Here is the caller graph for this function:



4.8.1.4 list_rm()

```
void list_rm (
            struct list * l,
            struct listentry * e )
```

Linked list implementation Linked list implementation for the part 3 of the project.

Removes an element from the list Set the attributes of the previous and next element of the element to be removed

Parameters

in	*l*	struct list* l -> the linked list whose element is to be removed
in	*e*	struct listentry* e -> the element to be removed

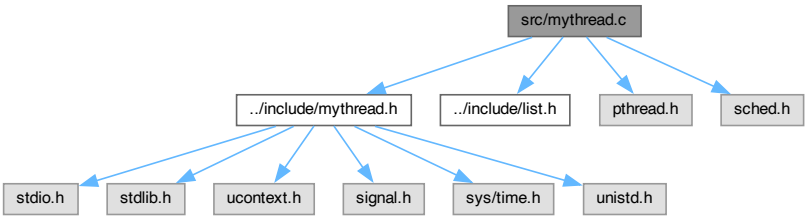
Returns

void

4.9 src/mythread.c File Reference

```
#include "../include/mythread.h"
#include "../include/list.h"
#include <pthread.h>
#include <sched.h>
```

Include dependency graph for mythread.c:



Functions

- void mythread_init ()
- void * mythread_create (void func(void *), void *arg)
- void mythread_yield ()
- void mythread_join ()
- struct lock * lock_new ()
- void lock_acquire (struct lock *lk)
- int lock_release (struct lock *lk)

Variables

- struct ucontext_t main_ctx
- struct list * l

4.9.1 Function Documentation

4.9.1.1 lock_acquire()

```
void lock_acquire (
            struct lock * lk )
```

Here is the caller graph for this function:

4.9.1.2 lock_new()

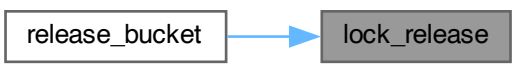
```
struct lock * lock_new ( )
```

Here is the caller graph for this function:

4.9.1.3 lock_release()

```
int lock_release (
            struct lock * lk )
```

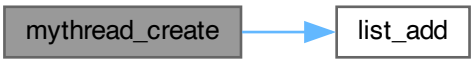
Here is the caller graph for this function:



4.9.1.4 mythread_create()

```
void * mythread_create (
            void  funcvoid *,
            void * arg )
```

Here is the call graph for this function:



4.9.1.5 mythread_init()

```
void mythread_init ( )
```

Here is the call graph for this function:

4.9.1.6 mythread_join()

```
void mythread_join ( )
```

4.9.1.7 mythread_yield()

```
void mythread_yield ( )
```

4.9.2 Variable Documentation

4.9.2.1 l

```
struct list* l
```

4.9.2.2 main_ctx

```
struct ucontext_t main_ctx
```

Index