

## Word Counter

Generated by Doxygen 1.9.1



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 hashmap_element_s Struct Reference	5
3.1.1 Member Data Documentation	5
3.1.1.1 data	5
3.1.1.2 key	5
3.2 hashmap_s Struct Reference	6
3.2.1 Member Data Documentation	6
3.2.1.1 lk	6
3.2.1.2 table	7
3.3 list Struct Reference	7
3.3.1 Member Data Documentation	7
3.3.1.1 head	7
3.3.1.2 tail	8
3.4 listentry Struct Reference	8
3.4.1 Member Data Documentation	8
3.4.1.1 data	8
3.4.1.2 next	9
3.4.1.3 prev	9
3.5 lock Struct Reference	9
3.5.1 Member Data Documentation	9
3.5.1.1 c	9
<b>4 File Documentation</b>	<b>11</b>
4.1 include/hm.h File Reference	11
4.1.1 Macro Definition Documentation	12
4.1.1.1 SZ	12
4.1.2 Function Documentation	12
4.1.2.1 acquire_bucket()	12
4.1.2.2 hashmap_create()	13
4.1.2.3 hashmap_get()	13
4.1.2.4 hashmap_iterator()	14
4.1.2.5 hashmap_put()	14
4.1.2.6 release_bucket()	15
4.2 include/list.h File Reference	16
4.2.1 Function Documentation	16
4.2.1.1 is_empty()	16
4.2.1.2 list_add()	17

4.2.1.3 list_new()	17
4.2.1.4 list_rm()	18
4.3 include/mythread.h File Reference	18
4.3.1 Function Documentation	19
4.3.1.1 lock_acquire()	20
4.3.1.2 lock_new()	20
4.3.1.3 lock_release()	20
4.3.1.4 mythread_create()	21
4.3.1.5 mythread_init()	21
4.3.1.6 mythread_join()	22
4.3.1.7 mythread_yield()	22
4.4 src/hm.c File Reference	22
4.4.1 Macro Definition Documentation	23
4.4.1.1 SZ	23
4.4.2 Function Documentation	23
4.4.2.1 acquire_bucket()	23
4.4.2.2 hashfn()	24
4.4.2.3 hashmap_create()	24
4.4.2.4 hashmap_get()	25
4.4.2.5 hashmap_iterator()	26
4.4.2.6 hashmap_put()	26
4.4.2.7 release_bucket()	26
4.5 src/list.c File Reference	27
4.5.1 Function Documentation	28
4.5.1.1 is_empty()	28
4.5.1.2 list_add()	28
4.5.1.3 list_new()	29
4.5.1.4 list_rm()	29
4.6 src/mythread.c File Reference	30
4.6.1 Function Documentation	30
4.6.1.1 lock_acquire()	31
4.6.1.2 lock_new()	31
4.6.1.3 lock_release()	31
4.6.1.4 mythread_create()	32
4.6.1.5 mythread_init()	32
4.6.1.6 mythread_join()	33
4.6.1.7 mythread_yield()	33
4.6.2 Variable Documentation	33
4.6.2.1 l	33
4.6.2.2 main_ctx	33

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">hashmap_element_s</a>	Elements consisting of key and value pairs that are to be stored in the Hashmap . . . . .	5
<a href="#">hashmap_s</a>	Hashmap consisting of linked lists with locks . . . . .	6
<a href="#">list</a>	Doubly linked list . . . . .	7
<a href="#">listentry</a>	Elements of the Linked List . . . . .	8
<a href="#">lock</a>	Lock object . . . . .	9



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">include/hm.h</a>	11
<a href="#">include/list.h</a>	16
<a href="#">include/mythread.h</a>	18
<a href="#">src/hm.c</a>	22
<a href="#">src/list.c</a>	27
<a href="#">src/mythread.c</a>	30





## Chapter 3

# Class Documentation

### 3.1 hashmap\_element\_s Struct Reference

Elements consisting of key and value pairs that are to be stored in the Hashmap.

```
#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 files:

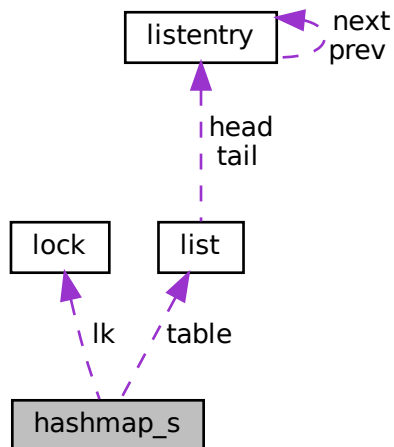
- include/[hm.h](#)
- src/[hm.c](#)

## 3.2 hashmap\_s Struct Reference

Hashmap consisting of linked lists with locks.

```
#include <hm.h>
```

Collaboration diagram for hashmap\_s:



### Public Attributes

- struct [list](#) \* [table](#) [[SZ](#)]  
*Array of linked list pointers.*
- struct [lock](#) \* [lk](#) [[SZ](#)]  
*Array of lock pointers corresponding to the linked lists of the hashtable.*

### 3.2.1 Member Data Documentation

#### 3.2.1.1 lk

```
struct lock * hashmap_s::lk
```

### 3.2.1.2 table

```
struct list * hashmap_s::table
```

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

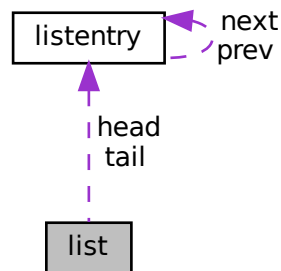
- include/hm.h
- src/hm.c

## 3.3 list Struct Reference

Doubly linked list.

```
#include <list.h>
```

Collaboration diagram for list:



### Public Attributes

- struct `listentry` \* `head`  
*Head of the linked list. Head is NULL when list is empty.*
- struct `listentry` \* `tail`  
*Tail of the linked list. Tail is NULL when list is empty.*

### 3.3.1 Member Data Documentation

#### 3.3.1.1 head

```
struct listentry * list::head
```

### 3.3.1.2 tail

```
struct listentry * list::tail
```

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

- include/list.h
- src/list.c

## 3.4 listentry Struct Reference

Elements of the Linked List.

```
#include <list.h>
```

Collaboration diagram for listentry:



### Public Attributes

- void \* [data](#)  
*Pointer of the data of the listentry.*
- struct [listentry](#) \* [prev](#)  
*Pointer of the previous element of the linked list.*
- struct [listentry](#) \* [next](#)  
*Pointer of the next element of the linked list.*

### 3.4.1 Member Data Documentation

#### 3.4.1.1 data

```
void * listentry::data
```

### 3.4.1.2 next

```
struct listentry * listentry::next
```

### 3.4.1.3 prev

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

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

- [include/list.h](#)
- [src/list.c](#)

## 3.5 lock Struct Reference

Lock object.

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

### Public Attributes

- `void * c`  
*Contains the thread which has acquired the lock.*

### 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

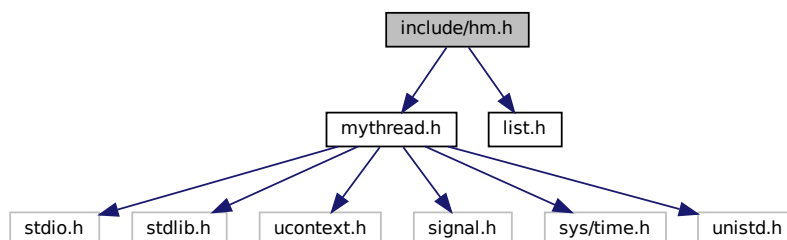
# File Documentation

### 4.1 include/hm.h File Reference

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

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

Include dependency graph for hm.h:



### Classes

- struct [hashmap\\_element\\_s](#)

*Elements consisting of key and value pairs that are to be stored in the Hashmap.*

- struct [hashmap\\_s](#)

*Hashmap consisting of linked lists with locks.*

### Macros

- `#define` [SZ](#) 4096

## Functions

- int [hashmap\\_create](#) (struct [hashmap\\_s](#) \*const out\_hashmap)  
*Initialises the hashmap.*
- int [hashmap\\_put](#) (struct [hashmap\\_s](#) \*const hashmap, const char \*key, void \*data)  
*Adds the key value pair to the hashmap.*
- void \* [hashmap\\_get](#) (struct [hashmap\\_s](#) \*const hashmap, const char \*key)  
*Removes and returns the value corresponding to the key in hashmap.*
- void [hashmap\\_iterator](#) (struct [hashmap\\_s](#) \*const hashmap, int(\*f)(struct [hashmap\\_element\\_s](#) \*const))  
*Executes argument function on each Key Value pair in the hashmap.*
- int [acquire\\_bucket](#) (struct [hashmap\\_s](#) \*const hashmap, const char \*key)  
*Acquire lock on a hashmap slot.*
- int [release\\_bucket](#) (struct [hashmap\\_s](#) \*const hashmap, const char \*key)  
*Releases the lock on acquired slot.*

### 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\(\)](#)

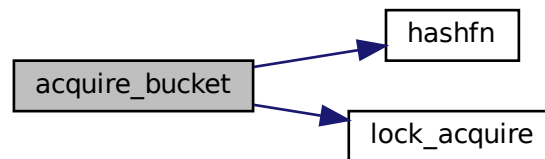
```
int acquire_bucket (  
    struct hashmap\_s *const hashmap,  
    const char * key )
```

##### Parameters

<i>hashmap</i>	: The pointer to the hashmap.
<i>key</i>	: The key for which the lock has to be set.



Here is the call graph for this function:



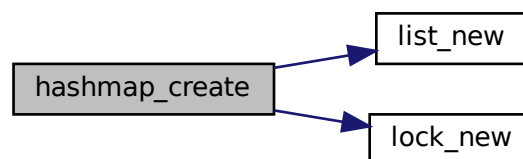
#### 4.1.2.2 hashmap\_create()

```
int hashmap_create (  
    struct hashmap\_s *const out_hashmap )
```

##### Parameters

<code>out_hashmap</code>	: The pointer to the hashmap.
--------------------------	-------------------------------

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 )
```

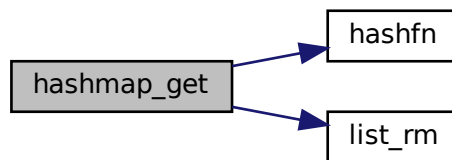
**Parameters**

<i>hashmap</i>	: The pointer to the hashmap.
<i>key</i>	: Key corresponding to which value is to be found.

**Returns**

Value corresponding to the key in argument (if key exists), NULL otherwise.

If the key exists in the hashmap then it returns the corresponding value of the key. Otherwise, returns NULL. 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 )
```

**Parameters**

<i>hashmap</i>	: The pointer to the hashmap.
<i>f</i>	: The function which is going to be executed on all of the key value pairs in the hashmap.

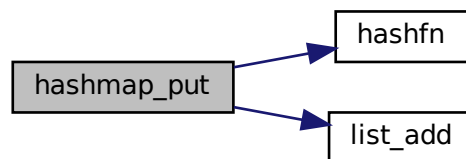
**4.1.2.5 hashmap\_put()**

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

## Parameters

<i>hashmap</i>	: The pointer to the hashmap.
<i>key</i>	: Key of the element to be added.
<i>data</i>	: Corresponding value of the key.

Here is the call graph for this function:



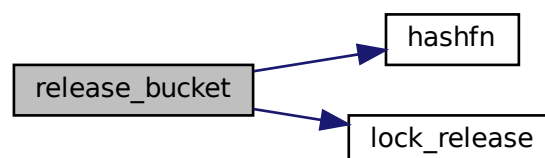
#### 4.1.2.6 release\_bucket()

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

## Parameters

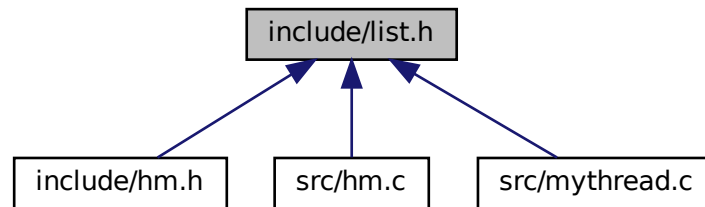
<i>hashmap</i>	: The pointer to the hashmap.
<i>key</i>	: The key for which the lock has to be removed.

Here is the call graph for this function:



## 4.2 include/list.h File Reference

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



### Classes

- struct [list](#)  
*Doubly linked list.*
- struct [listentry](#)  
*Elements of the Linked List.*

### Functions

- void [list\\_rm](#) (struct [list](#) \*l, struct [listentry](#) \*e)  
*Removes the given element from the Linked List.*
- struct [listentry](#) \* [list\\_add](#) (struct [list](#) \*l, void \*data)  
*Adds the given element to the Linked List.*
- struct [list](#) \* [list\\_new](#) ()
- int [is\\_empty](#) (struct [list](#) \*l)

### 4.2.1 Function Documentation

#### 4.2.1.1 is\_empty()

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

#### Parameters

<i>l</i>	: Pointer to a linked list.
----------	-----------------------------

### Returns

Returns 1 if List is empty and 0 otherwise.

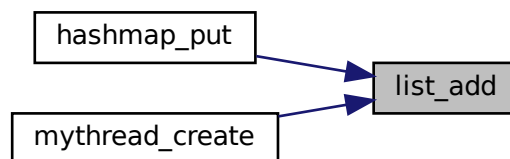
#### 4.2.1.2 list\_add()

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

### Parameters

<i>l</i>	: Pointer to the linked list.
<i>data</i>	: Element that needs to be added.

Creates an object of type listentry with its data as the given input and adds it to the end of the list. Here is the caller graph for this function:



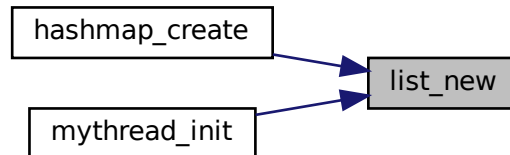
#### 4.2.1.3 list\_new()

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

**Returns**

Returns pointer to a dynamically allocated Linked List.

Here is the caller graph for this function:

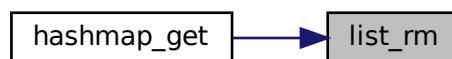
**4.2.1.4 list\_rm()**

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

**Parameters**

<i>l</i>	: pointer to the linked list.
<i>e</i>	: pointer to the linked list entry need to be removed.

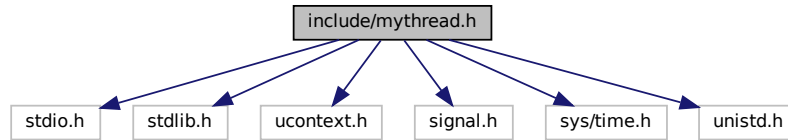
Removes the given element from the Linked List by iterating through it and removing when found. Here is the caller graph for this function:

**4.3 include/mythread.h File Reference**

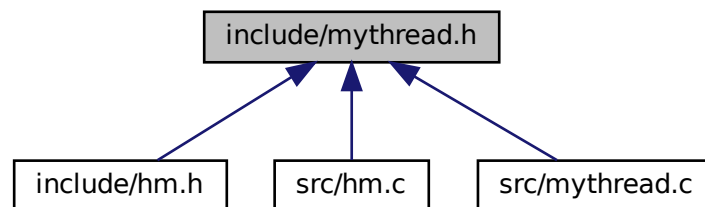
```
#include <stdio.h>
#include <stdlib.h>
#include <ucontext.h>
```

```
#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](#)  
*Lock object.*

## Functions

- void [mythread\\_init](#) ()
- void \* [mythread\\_create](#) (void func(void \*), void \*arg)  
*Creates a new thread.*
- void [mythread\\_join](#) ()
- void [mythread\\_yield](#) ()
- struct [lock](#) \* [lock\\_new](#) ()
- void [lock\\_acquire](#) (struct [lock](#) \*lk)
- int [lock\\_release](#) (struct [lock](#) \*lk)

### 4.3.1 Function Documentation

#### 4.3.1.1 lock\_acquire()

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

Sets lock for a key, and yields if the key is already locked. Here is the caller graph for this function:



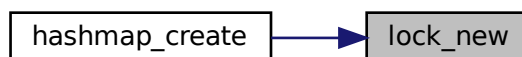
#### 4.3.1.2 lock\_new()

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

##### Returns

Returns a pointer to a dynamically allocated lock object.

Here is the caller graph for this function:

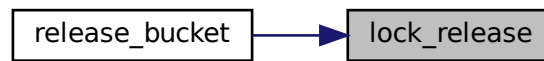


#### 4.3.1.3 lock\_release()

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



Releases lock for a key. Here is the caller graph for this function:



#### 4.3.1.4 mythread\_create()

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

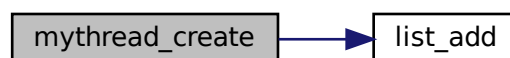
##### Parameters

<i>func</i>	: The function to be linked to the newly created thread.
<i>arg</i>	: The argument to be given when calling "func".

##### Returns

Returns the pointer to the context created.

Creates a new thread with the function `func` and adds it to the existing threadlist. Here is the call graph for this function:



#### 4.3.1.5 mythread\_init()

```
void mythread_init ( )
```

Initialises threadlist pointer to a dynamically allocated Linked List. Here is the call graph for this function:



#### 4.3.1.6 mythread\_join()

```
void mythread_join ( )
```

Waits for other threads to complete. Used in case of dependent threads.

#### 4.3.1.7 mythread\_yield()

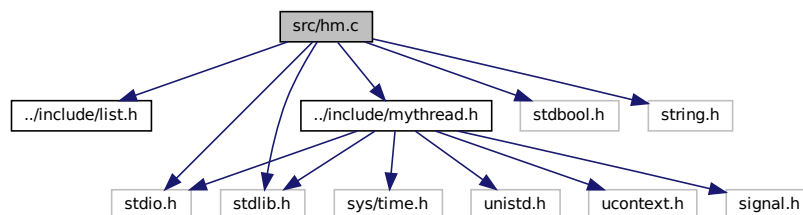
```
void mythread_yield ( )
```

Performs context switching. Used while yielding from one context to another without completing the current context.

## 4.4 src/hm.c File Reference

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

Include dependency graph for hm.c:



## Classes

- struct [hashmap\\_element\\_s](#)  
*Elements consisting of key and value pairs that are to be stored in the Hashmap.*
- struct [hashmap\\_s](#)  
*Hashmap consisting of linked lists with locks.*

## Macros

- `#define` [SZ](#) 4096

## Functions

- int [hashfn](#) (const char \*key)
- int [hashmap\\_create](#) (struct [hashmap\\_s](#) \*const out\_hashmap)  
*Initialises the hashmap.*
- int [hashmap\\_put](#) (struct [hashmap\\_s](#) \*const hashmap, const char \*key, void \*data)  
*Adds the key value pair to the hashmap.*
- void \* [hashmap\\_get](#) (struct [hashmap\\_s](#) \*const hashmap, const char \*key)  
*Removes and returns the value corresponding to the key in hashmap.*
- void [hashmap\\_iterator](#) (struct [hashmap\\_s](#) \*const hashmap, int(\*f)(struct [hashmap\\_element\\_s](#) \*const))  
*Executes argument function on each Key Value pair in the hashmap.*
- int [acquire\\_bucket](#) (struct [hashmap\\_s](#) \*const hashmap, const char \*key)  
*Acquire lock on a hashmap slot.*
- int [release\\_bucket](#) (struct [hashmap\\_s](#) \*const hashmap, const char \*key)  
*Releases the lock on acquired slot.*

### 4.4.1 Macro Definition Documentation

#### 4.4.1.1 SZ

```
#define SZ 4096
```

### 4.4.2 Function Documentation

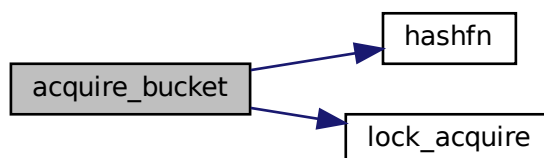
#### 4.4.2.1 acquire\_bucket()

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

**Parameters**

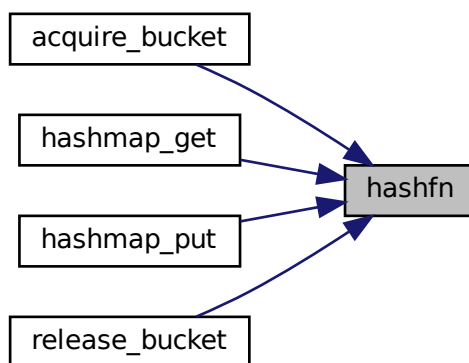
<i>hashmap</i>	: The pointer to the hashmap.
<i>key</i>	: The key for which the lock has to be set.

Here is the call graph for this function:

**4.4.2.2 hashfn()**

```
int hashfn (
    const char * key )
```

Here is the caller graph for this function:

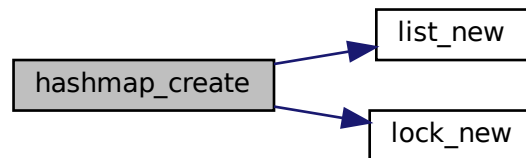
**4.4.2.3 hashmap\_create()**

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

## Parameters

<i>out_hashmap</i>	: The pointer to the hashmap.
--------------------	-------------------------------

Here is the call graph for this function:



#### 4.4.2.4 hashmap\_get()

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

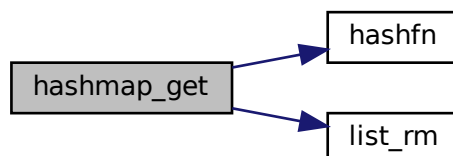
## Parameters

<i>hashmap</i>	: The pointer to the hashmap.
<i>key</i>	: Key corresponding to which value is to be found.

## Returns

Value corresponding to the key in argument (if key exists), NULL otherwise.

If the key exists in the hashmap then it returns the corresponding value of the key. Otherwise, returns NULL. Here is the call graph for this function:



#### 4.4.2.5 hashmap\_iterator()

```
void hashmap_iterator (
    struct hashmap\_s *const hashmap,
    int (*) (struct hashmap\_element\_s *const) f )
```

##### Parameters

<i>hashmap</i>	: The pointer to the hashmap.
<i>f</i>	: The function which is going to be executed on all of the key value pairs in the hashmap.

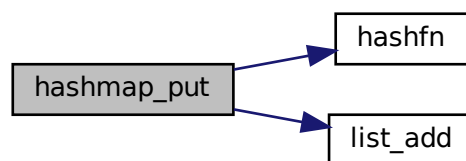
#### 4.4.2.6 hashmap\_put()

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

##### Parameters

<i>hashmap</i>	: The pointer to the hashmap.
<i>key</i>	: Key of the element to be added.
<i>data</i>	: Corresponding value of the key.

Here is the call graph for this function:



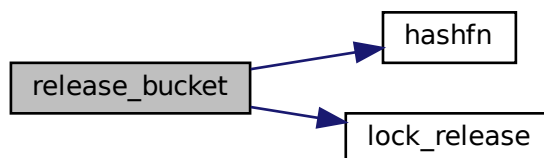
#### 4.4.2.7 release\_bucket()

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

## Parameters

<i>hashmap</i>	: The pointer to the hashmap.
<i>key</i>	: The key for which the lock has to be removed.

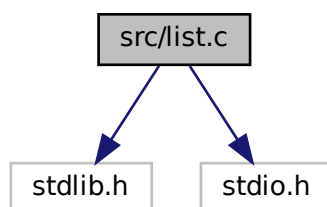
Here is the call graph for this function:



## 4.5 src/list.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
```

Include dependency graph for list.c:



## Classes

- struct [list](#)  
*Doubly linked list.*
- struct [listentry](#)  
*Elements of the Linked List.*

## Functions

- void `list_rm` (struct `list` \*`l`, struct `listentry` \*`e`)  
*Removes the given element from the Linked List.*
- struct `listentry` \* `list_add` (struct `list` \*`l`, void \*`data`)  
*Adds the given element to the Linked List.*
- struct `list` \* `list_new` ()
- int `is_empty` (struct `list` \*`l`)

### 4.5.1 Function Documentation

#### 4.5.1.1 `is_empty()`

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

##### Parameters

<code>l</code>	: Pointer to a linked list.
----------------	-----------------------------

##### Returns

Returns 1 if List is empty and 0 otherwise.

#### 4.5.1.2 `list_add()`

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

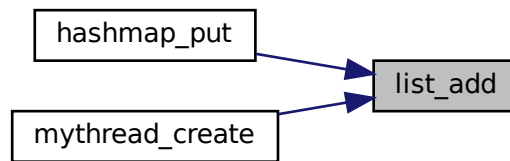
##### Parameters

<code>l</code>	: Pointer to the linked list.
<code>data</code>	: Element that needs to be added.

Creates an object of type `listentry` with its data as the given input and adds it to the end of the list. Here is the caller



graph for this function:



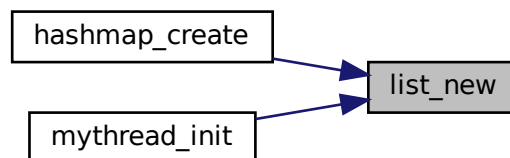
#### 4.5.1.3 list\_new()

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

##### Returns

Returns pointer to a dynamically allocated Linked List.

Here is the caller graph for this function:



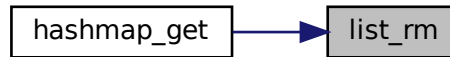
#### 4.5.1.4 list\_rm()

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

##### Parameters

<i>l</i>	: pointer to the linked list.
<i>e</i>	: pointer to the linked list entry need to be removed.

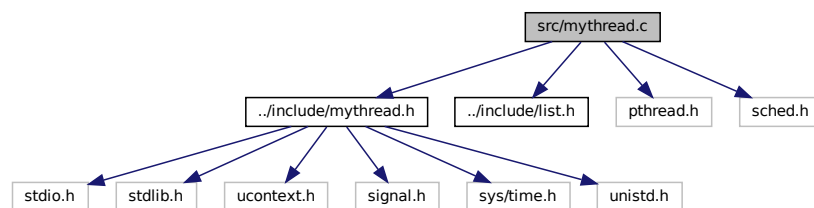
Removes the given element from the Linked List by iterating through it and removing when found. Here is the caller graph for this function:



## 4.6 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)  
*Creates a new thread.*
- 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`

### 4.6.1 Function Documentation

#### 4.6.1.1 lock\_acquire()

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

Sets lock for a key, and yields if the key is already locked. Here is the caller graph for this function:



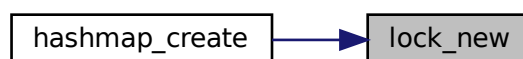
#### 4.6.1.2 lock\_new()

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

##### Returns

Returns a pointer to a dynamically allocated lock object.

Here is the caller graph for this function:



#### 4.6.1.3 lock\_release()

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

Releases lock for a key. Here is the caller graph for this function:



#### 4.6.1.4 mythread\_create()

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

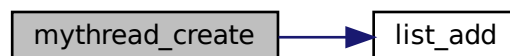
##### Parameters

<i>func</i>	: The function to be linked to the newly created thread.
<i>arg</i>	: The argument to be given when calling "func".

##### Returns

Returns the pointer to the context created.

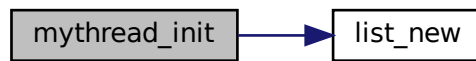
Creates a new thread with the function `func` and adds it to the existing threadlist. Here is the call graph for this function:



#### 4.6.1.5 mythread\_init()

```
void mythread_init ( )
```

Initialises threadlist pointer to a dynamically allocated Linked List. Here is the call graph for this function:



#### 4.6.1.6 mythread\_join()

```
void mythread_join ( )
```

Waits for other threads to complete. Used in case of dependent threads.

#### 4.6.1.7 mythread\_yield()

```
void mythread_yield ( )
```

Performs context switching. Used while yielding from one context to another without completing the current context.

### 4.6.2 Variable Documentation

#### 4.6.2.1 l

```
struct list* l
```

#### 4.6.2.2 main\_ctx

```
struct ucontext_t main_ctx
```



# Index

acquire\_bucket

hm.c, [23](#)

hm.h, [12](#)

c

lock, [9](#)

data

hashmap\_element\_s, [5](#)

listentry, [8](#)

hashfn

hm.c, [24](#)

hashmap\_create

hm.c, [24](#)

hm.h, [13](#)

hashmap\_element\_s, [5](#)

data, [5](#)

key, [5](#)

hashmap\_get

hm.c, [25](#)

hm.h, [13](#)

hashmap\_iterator

hm.c, [25](#)

hm.h, [14](#)

hashmap\_put

hm.c, [26](#)

hm.h, [14](#)

hashmap\_s, [6](#)

lk, [6](#)

table, [6](#)

head

list, [7](#)

hm.c

acquire\_bucket, [23](#)

hashfn, [24](#)

hashmap\_create, [24](#)

hashmap\_get, [25](#)

hashmap\_iterator, [25](#)

hashmap\_put, [26](#)

release\_bucket, [26](#)

SZ, [23](#)

hm.h

acquire\_bucket, [12](#)

hashmap\_create, [13](#)

hashmap\_get, [13](#)

hashmap\_iterator, [14](#)

hashmap\_put, [14](#)

release\_bucket, [15](#)

SZ, [12](#)

include/hm.h, [11](#)

include/list.h, [16](#)

include/mythread.h, [18](#)

is\_empty

list.c, [28](#)

list.h, [16](#)

key

hashmap\_element\_s, [5](#)

l

mythread.c, [33](#)

list, [7](#)

head, [7](#)

tail, [7](#)

list.c

is\_empty, [28](#)

list\_add, [28](#)

list\_new, [29](#)

list\_rm, [29](#)

list.h

is\_empty, [16](#)

list\_add, [17](#)

list\_new, [17](#)

list\_rm, [18](#)

list\_add

list.c, [28](#)

list.h, [17](#)

list\_new

list.c, [29](#)

list.h, [17](#)

list\_rm

list.c, [29](#)

list.h, [18](#)

listentry, [8](#)

data, [8](#)

next, [8](#)

prev, [9](#)

lk

hashmap\_s, [6](#)

lock, [9](#)

c, [9](#)

lock\_acquire

mythread.c, [30](#)

mythread.h, [19](#)

lock\_new

mythread.c, [31](#)

mythread.h, [20](#)

lock\_release

mythread.c, [31](#)

- mythread.h, [20](#)
- main\_ctx
  - mythread.c, [33](#)
- mythread.c
  - l, [33](#)
  - lock\_acquire, [30](#)
  - lock\_new, [31](#)
  - lock\_release, [31](#)
  - main\_ctx, [33](#)
  - mythread\_create, [32](#)
  - mythread\_init, [32](#)
  - mythread\_join, [33](#)
  - mythread\_yield, [33](#)
- mythread.h
  - lock\_acquire, [19](#)
  - lock\_new, [20](#)
  - lock\_release, [20](#)
  - mythread\_create, [21](#)
  - mythread\_init, [21](#)
  - mythread\_join, [22](#)
  - mythread\_yield, [22](#)
- mythread\_create
  - mythread.c, [32](#)
  - mythread.h, [21](#)
- mythread\_init
  - mythread.c, [32](#)
  - mythread.h, [21](#)
- mythread\_join
  - mythread.c, [33](#)
  - mythread.h, [22](#)
- mythread\_yield
  - mythread.c, [33](#)
  - mythread.h, [22](#)
- next
  - listentry, [8](#)
- prev
  - listentry, [9](#)
- release\_bucket
  - hm.c, [26](#)
  - hm.h, [15](#)
- src/hm.c, [22](#)
- src/list.c, [27](#)
- src/mythread.c, [30](#)
- SZ
  - hm.c, [23](#)
  - hm.h, [12](#)
- table
  - hashmap\_s, [6](#)
- tail
  - list, [7](#)