Word Counter

Generated by Doxygen 1.9.1

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

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 . . . 6

3.3 list Struct Reference . . . 6

3.3.1 Member Data Documentation . . . 7

3.3.1.1 head . . . 7

3.3.1.2 tail . . . 7

3.4 listentry Struct Reference . . . 7

3.4.1 Member Data Documentation . . . 7

3.4.1.1 data . . . 8

3.4.1.2 next . . . 8

3.4.1.3 prev . . . 8

3.5 lock Struct Reference . . . 8

3.5.1 Member Data Documentation . . . 8

3.5.1.1 c . . . 8

4 File Documentation 9

4.1 include/hm.h File Reference . . . 9

4.1.1 Macro Definition Documentation . . . 10

4.1.1.1 SZ . . . 10

4.1.2 Function Documentation . . . 10

4.1.2.1 acquire_bucket() . . . 10

4.1.2.2 hashmap_create() . . . 10

4.1.2.3 hashmap_get() . . . 10

4.1.2.4 hashmap_iterator() . . . 11

4.1.2.5 hashmap_put() . . . 11

4.1.2.6 release_bucket() . . . 11

4.2 include/list.h File Reference . . . 12

4.2.1 Function Documentation . . . 12

4.2.1.1 is_empty() . . . 12

4.2.1.2 list_add() . . . 12

4.2.1.3 list_new() 13
4.2.1.4 list_rm() 13
4.3 include/mythread.h File Reference 13
4.3.1 Function Documentation 14
4.3.1.1 lock_acquire() 14
4.3.1.2 lock_new() 14
4.3.1.3 lock_release() 14
4.3.1.4 mythread_create() 14
4.3.1.5 mythread_init() 15
4.3.1.6 mythread_join() 15
4.3.1.7 mythread_yield() 15
4.4 src/hm.c File Reference 15
4.4.1 Macro Definition Documentation 16
4.4.1.1 SZ 16
4.4.2 Function Documentation 16
4.4.2.1 acquire_bucket() 16
4.4.2.2 hashfn() 17
4.4.2.3 hashmap_create() 17
4.4.2.4 hashmap_get() 17
4.4.2.5 hashmap_iterator() 17
4.4.2.6 hashmap_put() 18
4.4.2.7 release_bucket() 18
4.5 src/list.c File Reference 18
4.5.1 Function Documentation 19
4.5.1.1 is_empty() 19
4.5.1.2 list_add() 19
4.5.1.3 list_new() 19
4.5.1.4 list_rm() 20
4.6 src/mythread.c File Reference 20
4.6.1 Function Documentation 20
4.6.1.1 lock_acquire() 21
4.6.1.2 lock_new() 21
4.6.1.3 lock_release() 21
4.6.1.4 mythread_create() 21
4.6.1.5 mythread_init() 21
4.6.1.6 mythread_join() 22
4.6.1.7 mythread_yield() 22
4.6.2 Variable Documentation 22
4.6.2.1 l 22
4.6.2.2 main_ctx 22

Index **23**

Chapter 1

Class Index

1.1 Class List

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

hashmap_element_s
Elements consisting of key and value pairs that are to be stored in the Hashmap 5
hashmap_s
Hashmap consisting of linked lists with locks 6
list
Doubly linked list 6
listentry
Elements of the Linked List 7
lock
Lock object 8

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

include/hm.h 9
include/list.h 12
include/mythread.h 13
src/hm.c 15
src/list.c 18
src/mythread.c 20

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

hashmap	: The pointer to the hashmap.
key	: 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

out_hashmap	: 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

hashmap	: The pointer to the hashmap.
key	: 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

hashmap	: The pointer to the hashmap.
f	: 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

hashmap	: The pointer to the hashmap.
key	: Key of the element to be added.
data	: 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

hashmap	: The pointer to the hashmap.
key	: 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

l	: 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

l	: Pointer to the linked list.
data	: 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

l	: pointer to the linked list.
e	: 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

func	: The function to be linked to the newly created thread.
arg	: 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

hashmap	: The pointer to the hashmap.
key	: 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

out_hashmap	: 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

hashmap	: The pointer to the hashmap.
key	: 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

hashmap	: The pointer to the hashmap.
f	: 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

hashmap	: The pointer to the hashmap.
key	: Key of the element to be added.
data	: 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

hashmap	: The pointer to the hashmap.
key	: 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

l	: 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

l	: Pointer to the linked list.
data	: 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

l	: pointer to the linked list.
e	: 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

func	: The function to be linked to the newly created thread.
arg	: 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, 16
    hm.h, 10

c
    lock, 8

data
    hashmap_element_s, 5
    listentry, 7

hashfn
    hm.c, 16
hashmap_create
    hm.c, 17
    hm.h, 10
hashmap_element_s, 5
    data, 5
    key, 5
hashmap_get
    hm.c, 17
    hm.h, 10
hashmap_iterator
    hm.c, 17
    hm.h, 11
hashmap_put
    hm.c, 18
    hm.h, 11
hashmap_s, 6
    lk, 6
    table, 6
head
    list, 7
hm.c
    acquire_bucket, 16
    hashfn, 16
    hashmap_create, 17
    hashmap_get, 17
    hashmap_iterator, 17
    hashmap_put, 18
    release_bucket, 18
    SZ, 16
hm.h
    acquire_bucket, 10
    hashmap_create, 10
    hashmap_get, 10
    hashmap_iterator, 11
    hashmap_put, 11
    release_bucket, 11
    SZ, 10

include/hm.h, 9
include/list.h, 12
include/mythread.h, 13
is_empty
    list.c, 19
    list.h, 12

key
    hashmap_element_s, 5

l
    mythread.c, 22
list, 6
    head, 7
    tail, 7
list.c
    is_empty, 19
    list_add, 19
    list_new, 19
    list_rm, 20
list.h
    is_empty, 12
    list_add, 12
    list_new, 13
    list_rm, 13
list_add
    list.c, 19
    list.h, 12
list_new
    list.c, 19
    list.h, 13
list_rm
    list.c, 20
    list.h, 13
listentry, 7
    data, 7
    next, 8
    prev, 8
lk
    hashmap_s, 6
lock, 8
    c, 8
lock_acquire
    mythread.c, 20
    mythread.h, 14
lock_new
    mythread.c, 21
    mythread.h, 14
lock_release
    mythread.c, 21

mythread.h, 14

main_ctx
    mythread.c, 22
mythread.c
    l, 22
    lock_acquire, 20
    lock_new, 21
    lock_release, 21
    main_ctx, 22
    mythread_create, 21
    mythread_init, 21
    mythread_join, 21
    mythread_yield, 22
mythread.h
    lock_acquire, 14
    lock_new, 14
    lock_release, 14
    mythread_create, 14
    mythread_init, 15
    mythread_join, 15
    mythread_yield, 15
mythread_create
    mythread.c, 21
    mythread.h, 14
mythread_init
    mythread.c, 21
    mythread.h, 15
mythread_join
    mythread.c, 21
    mythread.h, 15
mythread_yield
    mythread.c, 22
    mythread.h, 15

next
    listentry, 8

prev
    listentry, 8

release_bucket
    hm.c, 18
    hm.h, 11

src/hm.c, 15
src/list.c, 18
src/mythread.c, 20
SZ
    hm.c, 16
    hm.h, 10

table
    hashmap_s, 6
tail
    list, 7