Proxy(c)2012

Generato da Doxygen 1.7.3

Fri Feb 17 2012 14:42:44

Indice

Capitolo 1

Indice delle strutture dati

1.1 Strutture dati

Queste sono le strutture dati con una loro breve descrizione:

Capitolo 2

Indice dei file

2.1 Elenco dei file

Questo è un elenco di tutti i file con una loro breve descrizione:

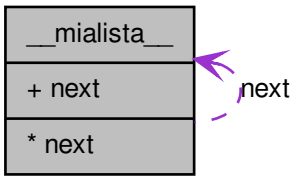
Capitolo 3

Documentazione delle classi

3.1 Riferimenti per la struct __mialista__

`#include <types.h>`

Diagramma di collaborazione per __mialista__:



Campi

- struct __mialista__ * next

3.1.1 Documentazione dei campi

3.1.1.1 struct __mialista__ * __mialista__::next

Puntatore al successivo

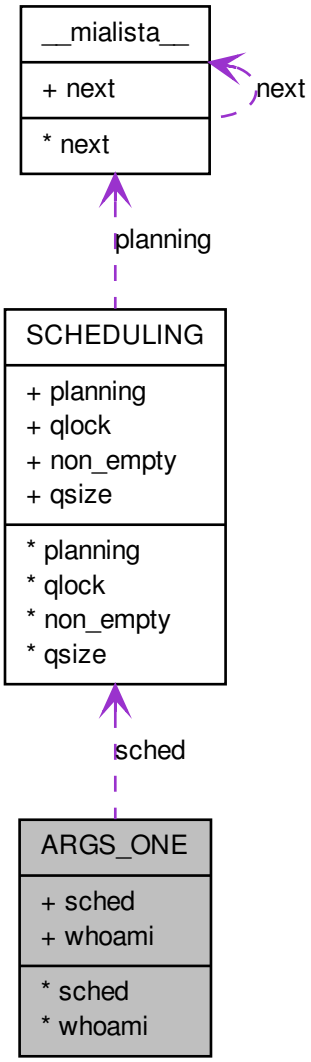
La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.2 Riferimenti per la struct ARGS_ONE

`#include <types.h>`

Diagramma di collaborazione per ARGS_ONE:



Campi

- SCHEDULING ∗ sched
- char whoami

3.2.1 Descrizione dettagliata

ARGS_ONE Questa struttura dati consente di specificare, ai thread che usano server-Pool, a quale indice di cacheRes devono far riferimento.

3.2.2 Documentazione dei campi

3.2.2.1 SCHEDULING∗ ARGS_ONE::sched

Puntatore allo scheduling associato

3.2.2.2 char ARGS_ONE::whoami

Identificatore numerico del thread

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.3 Riferimenti per la struct connection

```
#include <types.h>
```

Campi

- int fd_client
- int fd_server
- int tentativi
- int expiry
- char PATH [MAX_PATH]
- char sIP [16]
- char sPort [7]

3.3.1 Descrizione dettagliata

struct connection: Struttura dati contenente lo stato attuale della connessione

3.3.2 Documentazione dei campi

3.3.2.1 int connection::expiry

Tempo di expire associata alla risorsa in download

3.3.2.2 int connection::fd_client

Eventuale File Descriptor del client richiedente

3.3.2.3 int connection::fd_server

Eventuale File Descriptor del server al quale si richiede

3.3.2.4 char connection::PATH[MAX_PATH]

Stringa indicante il percorso della richiesta

3.3.2.5 char connection::sIP[16]

Indirizzo IP del server al quale si inoltra la richiesta

3.3.2.6 char connection::sPort[7]

Porta del server al quale si inoltra la richiesta

3.3.2.7 int connection::tentativi

Conteggio dei tentativi effettuati

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.4 Riferimenti per la struct CS_struct

```
#include <lett_scritt.h>
```

Campi

- pthread_mutex_t attr

3.4.1 Documentazione dei campi

3.4.1.1 pthread_mutex_t CS_struct::attr

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/hashtable/include/lett_scritt.h

3.5 Riferimenti per la struct hash_tbl

```
#include <hash.h>
```

Campi

- unsigned int elements
- unsigned int value

3.5.1 Descrizione dettagliata

struct hash_tbl: Struttura dati di hashing

3.5.2 Documentazione dei campi

3.5.2.1 unsigned int hash_tbl::elements

3.5.2.2 unsigned int hash_tbl::value

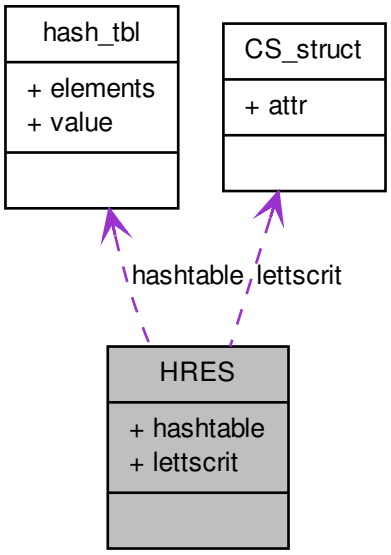
La documentazione per questa struct è stata generata a partire dal seguente file:

- src/hashtable/include/hash.h

3.6 Riferimenti per la struct HRES

```
#include <hres.h>
```

Diagramma di collaborazione per HRES:



Campi

- home giacomo Scrivania connect_version connect_paul src hashtable include hres h home giacomo Scrivania connect_version connect_paul src hashtable include hres h home giacomo Scrivania connect_version connect_paul src hashtable include hres h struct hash_tbl hashtable [MAX_HRES]
- struct CS_struct lettscrit [MAX_HRES]

3.6.1 Documentazione dei campi

3.6.1.1 home giacomo Scrivania connect_version connect_paul src hashtable include hres h home giacomo Scrivania connect_version connect_paul src hashtable include hres h home giacomo Scrivania connect_version connect_paul src hashtable include hres h struct hash_tbl HRES::hashtable[MAX_HRES]

3.6.1.2 struct CS_struct HRES::lettscrit[MAX_HRES]

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/hashtable/include/hres.h

3.7 Riferimenti per la struct JOB

```
#include <types.h>
```

Campi

- void(∗ procedura)()
- void ∗ args

3.7.1 Descrizione dettagliata

JOB: Questa struttura dati contiene il lavoro che deve essere effettuato dal thread in questione. Il thread in particolare dovrà eseguire una data funzione, dove è presente una nostra procedura, con degli argomenti che verranno passati successivamente alla sua chiamata

3.7.2 Documentazione dei campi

3.7.2.1 void∗ JOB::args

Argomenti da passare alla procedura

3.7.2.2 void(∗ JOB::procedura)()

Procedura associata alla richiesta

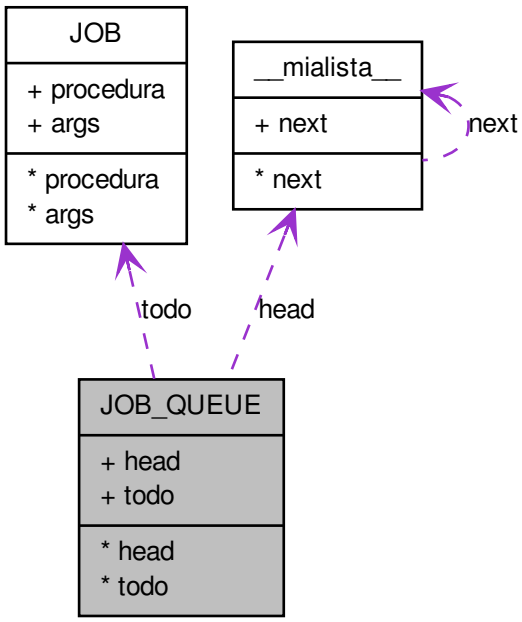
La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.8 Riferimenti per la struct JOB_QUEUE

```
#include <types.h>
```

Diagramma di collaborazione per JOB_QUEUE:



Campi

- LIST head
- JOB todo

3.8.1 Descrizione dettagliata

JOB_QUEUE: Questa struttura dati serve per implementare una coda di lavori da effettuare, che verranno messi all'interno della struttura dati in questioneRetorica_musicale

3.8.2 Documentazione dei campi

3.8.2.1 LIST JOB_QUEUE::head

Testa della lista

3.8.2.2 JOB JOB_QUEUE::todo

Lavoro da effettuare

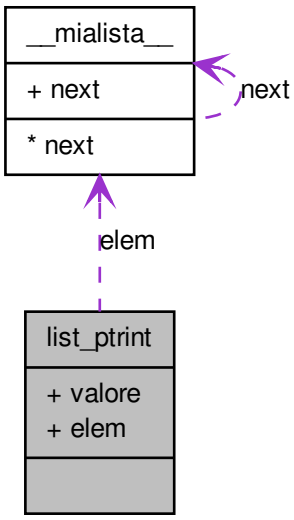
La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.9 Riferimenti per la struct list_ptrint

`#include <lists.h>`

Diagramma di collaborazione per list_ptrint:



Campi

- void * valore
- LIST elem

3.9.1 Documentazione dei campi

3.9.1.1 LIST list_ptrint::elem

3.9.1.2 void* list_ptrint::valore

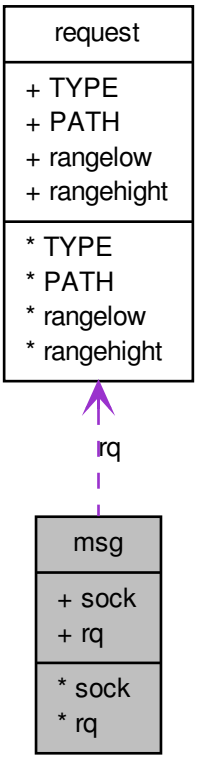
La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/lists.h

3.10 Riferimenti per la struct msg

```
#include <types.h>
```

Diagramma di collaborazione per msg:



Campi

- int sock
- request rq

3.10.1 Descrizione dettagliata

msg: Struttura per il passaggio dei parametri alle funzioni di gestione delle pool.

3.10.2 Documentazione dei campi

3.10.2.1 request msg::rq

la richiesta(parsata) fatta dal client

3.10.2.2 int msg::sock

il fd che rappresenta la connessione col client

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.11 Riferimenti per la struct proxy

```
#include <types.h>
```

Campi

- struct sockaddr_in meStesso
- int fd
- int level
- in_addr_t ip
- int port

3.11.1 Descrizione dettagliata

struct proxy: Struttura dati dove sono raccolte le informazioni per la connessione lato client

3.11.2 Documentazione dei campi

3.11.2.1 int proxy::fd

Fd del proxy lato client

3.11.2.2 in_addr_t proxy::ip

IP del proxy

3.11.2.3 int proxy::level

Livello di prefetch

3.11.2.4 struct sockaddr_in proxy::meStesso

3.11.2.5 int proxy::port

porta del proxy

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.12 Riferimenti per la struct request

```
#include <types.h>
```

Campi

- char TYPE
- char PATH [MAX_PATH]
- unsigned int rangelow
- unsigned int rangehight

3.12.1 Descrizione dettagliata

request: Struttura dati per la gestione della richiesta lato client

3.12.2 Documentazione dei campi

3.12.2.1 char request::PATH[MAX_PATH]

Percorso della risorsa

3.12.2.2 unsigned int request::rangehight

Eventuale higher-range

3.12.2.3 unsigned int request::rangelow

Eventuale lower-rage

3.12.2.4 char request::TYPE

Tipo di richiesta effettuata

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.13 Riferimenti per la struct request_cache

`#include <toclient.h>`

Campi

- int fd_client
- int fd_file
- char * elemen
- char remote [MAX_FILE]

3.13.1 Documentazione dei campi

3.13.1.1 char* request_cache::elemen

3.13.1.2 int request_cache::fd_client

3.13.1.3 int request_cache::fd_file

3.13.1.4 char request_cache::remote[MAX_FILE]

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/toclient.h

3.14 Riferimenti per la struct reslist

`#include <types.h>`

Campi

- int prefetch_num
- char * remote_path

3.14.1 Descrizione dettagliata

reslist: Struttura per il passaggio dei parametri delle richieste di prefetch, inesistenti in cache

3.14.2 Documentazione dei campi

3.14.2.1 int reslist::prefetch_num

Livello attuale di prefetching

3.14.2.2 char* reslist::remote_path

Stringa indicante la risorsa da ottenere

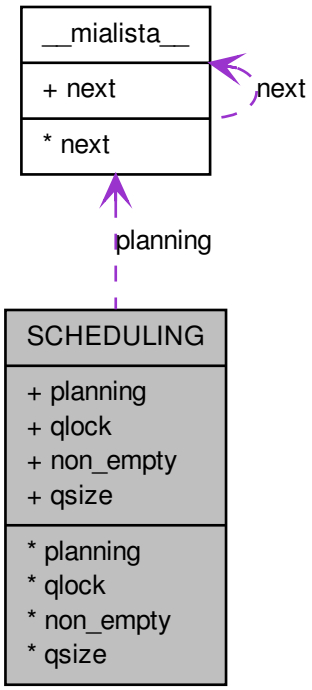
La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

3.15 Riferimenti per la struct SCHEDULING

```
#include <types.h>
```

Diagramma di collaborazione per SCHEDULING:



Campi

- LIST planning
- pthread_mutex_t qlock
- pthread_cond_t non_empty
- unsigned short int qsize

3.15.1 Descrizione dettagliata

SCHEDULING: Questa struttura dati serve per ricordarsi, all'interno del programma, lo stato corrente dell'esecuzione dei lavori all'interno del proxY.

3.15.2 Documentazione dei campi

3.15.2.1 pthread_cond_t SCHEDULING::non_empty

Condizione di coda non vuota

3.15.2.2 LIST SCHEDULING::planning

Lista dei JOB che devono essere svolti dai thread correnti

3.15.2.3 pthread_mutex_t SCHEDULING::qlock

Lock per l'accesso alla lista

3.15.2.4 unsigned short int SCHEDULING::qsize

Dimensione della coda dei dati

La documentazione per questa struct è stata generata a partire dal seguente file:

- src/proxy/include/types.h

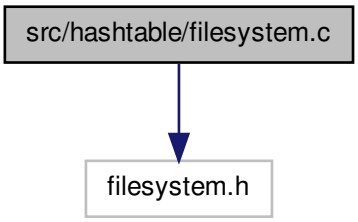
Capitolo 4

Documentazione dei file

4.1 Riferimenti per il file src/hashtable/filesystem.c

```
#include "filesystem.h"
```

Grafo delle dipendenze di inclusione per filesystem.c:



Funzioni

- void recursiveDelete (char *dirname)
- char file_exists (char *filename, unsigned int *time, char *is_blank, struct stat *retbuf, int *pfd)
- char folder_empty (char *dirname)
- char * obtain_local (char *remote)
- char * resource_exists (char *remote, int deallocate, char *result, unsigned int *time, char *blank, struct stat *retbuf, int *pfd)
- void re_new_resource (char *cache_path, struct stat *retbuf, int *pfd)
- void resource_remove (char *path)

4.1.1 Documentazione delle funzioni

4.1.1.1 char file_exists (char * *filename*, unsigned int * *time*, char * *is_blank*, struct stat * *retbuf*, int * *pfd*)

Funzioni accessorie per la cache file_exist(): Verifica l'esistenza del file *filename*. Se il pathname esiste ed è un file, restituisce 1, se esso è una cartella restituisce 2, se non esiste 0.

Parametri

filename,:	file locale del quale controllare l'esistenza
time,:	Conterrà il tempo da Epoch della creazione del file
is_blank,:	Restituisce, se passato come argomento, se il file è solamente troncato
retbuf,:	Restituisce lo fstat del file, se esiste.
pfd,:	Possibile (o NULL) puntatore al FileDescriptor: in quel caso il file non verrà chiuso

4.1.1.2 char folder_empty (char * *dirname*)

4.1.1.3 char* obtain_local (char * *remote*)

Funzioni fondamentali obtain_local(): Dato il pathning remoto, alloca in una stringa l'eventuale pathning locale corrispondente

Parametri

remote,:	percorso remoto del quale ottenere la conversione alla stringa corrente

4.1.1.4 void re_new_resource (char * *cache_path*, struct stat * *retbuf*, int * *pfd*)

re_new_resource(): Data il path della cache, crea la nuova risorsa: se le cartelle intermedie non esistono, esse vengono create, ed il file finale viene troncato ed aperto alla dimensione base massima. Restituisce la data di creazione del file

Parametri

cache_path,:	Identifica il percorso locale della risorsa
retbuf,:	Identifica lo stat del file che è stato aperto. Se non NULL viene aggiornato.
pfd,:	Identifica il FileDescriptor del nuovo file. Se non NULL viene aggiornato, altrimenti viene chiuso.

4.1.1.5 void recursiveDelete (char * *dirname*)

recursiveDelete(): Effettua l'eliminazione ricorsiva degli elementi all'interno della directory

Parametri

dirname,:	Percorso della cartella da svuotare ricorsivamente

4.1.1.6 char* resource_exists (char * *remote,* int *deallocate,* char * *result,* unsigned int * *time,* char * *blank,* struct stat * *retbuf,* int * *pfd*)

resource_exists(): Questa funzione verifica se il path remoto esista o meno. In base al valore del parametro *deallocate*, viene restituita o meno la stringa convertita della risorsa locale corrispondente

Parametri

remote,:	path remoto
deallocate,:	indica se deallocare l'indirizzo locale della risorsa corrispondente
result,:	verrà scritto in questo puntatore l'esistenza o meno del file
time,:	tempo di creazione della risorsa
blank,:	restituisce se il file è stato troncato o meno (e quindi vuoto)
retbuf,:	restituisce (se non NULL) il puntatore all'fstat del file in cache
pfd,:	Possibile (o NULL) puntatore al FileDescriptor: viene passato un fd aperto.

4.1.1.7 void resource_remove (char * *path*)

resource_remove(): Questa funzione effettua l'unlink sul file definito, ed inoltre via via eventualmente cancella le cartelle genitori se queste sono vuote.

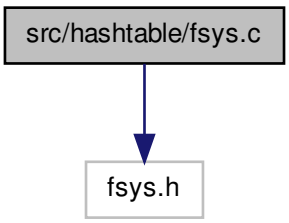
Parametri

path,:	Risorsa remota

4.2 Riferimenti per il file src/hashtable/fsys.c

```
#include "fsys.h"
```

Grafo delle dipendenze di inclusione per fsys.c:



Funzioni

- int FileCreate (char *remote_resource, void *file)
- int FileOpen (char *remote_resource, void **update)
- int FileClose (void *update, int fd)

4.2.1 Documentazione delle funzioni

4.2.1.1 int FileClose (void * *update*, int *fd*)

FileClose(): Effettua la chiusura del file mappato in memoria

Parametri

update,:	Puntatore alla risorsa mappata in memoria
fd,:	File descriptor della risorsa associata

Restituisce

Restituisce 0 in caso di insuccesso, altrimenti 1

4.2.1.2 int FileCreate (char * *remote_resource*, void * *file*)

FileCreate(): Questa funzione effettua la creazione del file, dato il nome della risorsa remota.

Parametri

remote_-resoruce,:	risorsa remota
expiry,:	tempo di expire
buffer,:	contenuto dell'informazione

Restituisce

restituisce 0 in caso di insuccesso, altrimenti 1

4.2.1.3 int FileOpen (char * *remote_resource,* void ** *update*)

FileOpen(): Questa funzione effettua l'apertura di un file che si assume non essere scaduto.

Parametri

remote_-resource,:	indicazione della risorsa remota
update,:	Indicazione dell'area di allocazione del file in cache

Restituisce

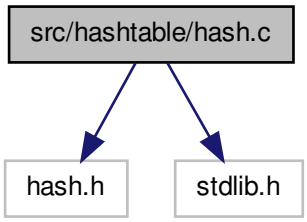
Restituisce 0 in caso di insuccesso, altrimenti il valore del filedescriptor

update = mmap(0,MAX_FILE,PROT_READ,MAP_SHARED,fd,0);

4.3 Riferimenti per il file src/hashtable/hash.c

```
#include "hash.h"
#include <stdlib.h>
```

Grafo delle dipendenze di inclusione per hash.c:



Definizioni

- #define SEED 13

Funzioni

- void init_hashtbl (struct hash_tbl ∗p)
- unsigned int hashb (char ∗p, unsigned int MULT, unsigned int NHASH)
- unsigned int hash (char ∗p)
- int hash_exists (struct hash_tbl ∗tbl, char ∗p)
- void hash_insert (struct hash_tbl ∗tbl, char ∗p)
- void hash_remove (struct hash_tbl ∗tbl, char ∗p)
- int hash_occupied (struct hash_tbl ∗tbl)

4.3.1 Documentazione delle definizioni

4.3.1.1 #define SEED 13

4.3.2 Documentazione delle funzioni

4.3.2.1 unsigned int hash (char ∗ *p*)

hash(): Funzione di hash

4.3.2.2 int hash_exists (struct hash_tbl ∗ *tbl*, char ∗ *p*)

hash_exists(): Verifica che una stringa sia presente all'interno della tavola

4.3.2.3 void hash_insert (struct hash_tbl ∗ *tbl*, char ∗ *p*)

hash_insert(): Effettua l'inserimento con collisioni

4.3.2.4 int hash_occupied (struct hash_tbl ∗ *tbl*)

hash_occupied(): Verifica se l'elemento è occupato

4.3.2.5 void hash_remove (struct hash_tbl ∗ *tbl*, char ∗ *p*)

hash_remove(): Effettua la rimozione di un elemento o di una collisione

4.3.2.6 unsigned int hashb (char ∗ *p*, unsigned int *MULT*, unsigned int *NHASH*)

hashb(): Funzione di base per lo hashing

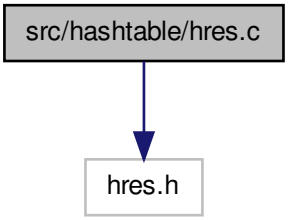
4.3.2.7 void init_hashtbl (struct hash_tbl ∗ *p*)

init_hashtbl(): Inizializzazione della hashtable

4.4 Riferimenti per il file src/hashtable/hres.c

```
#include "hres.h"
```

Grafo delle dipendenze di inclusione per hres.c:



Funzioni

- int parseHead_time (char ∗buf)
- void init_hres (HRES ∗ptr)
- int is_inhash (HRES ∗phash, char ∗remote_file)
- int handle_file_filesystem (HRES ∗phash, char ∗remote_file, void ∗res, int creat_oth_delete)
- int map_cached_file (HRES ∗hash, char ∗remote_file, void ∗∗toret)
- void close_cached_file (HRES ∗phash, char ∗remote_file, void ∗toclose, int fd)

4.4.1 Documentazione delle funzioni

4.4.1.1 void close_cached_file (HRES ∗ *phash,* char ∗ *remote_file,* void ∗ *toclose,* int *fd*)

4.4.1.2 int handle_file_filesystem (HRES ∗ *phash,* char ∗ *remote_file,* void ∗ *res,* int *creat_oth_delete*)

handle_file_filesystem(): Funzione per la gestione controllata degli elementi del file-system

Parametri

hash,:	Struttura dati di hashing
remote_file,:	Indicazione del file remoto di risorsa
res,:	Indica la risorsa contentente le informazioni, non utilizzata per l'eliminazione del file
creat_oth_-delete,:	Indica se il valore è maggiore di zero, di creare il file, altrimenti di cancellarlo.

Restituisce

La funzione restituisce comunque -1 in caso di cancellazione, la posizione nella htable (0 based) in caso di creazione con successo, altrimenti -1

4.4.1.3 void init_hres (HRES * *ptr*)

init_hres(): Inizializzazione della struttura dati

4.4.1.4 int is_inhash (HRES * *phash,* char * *remote_file*)

is_inhash(): Controlla se il file è presente in cache, e se esiste effettivamente all'interno del filesystem

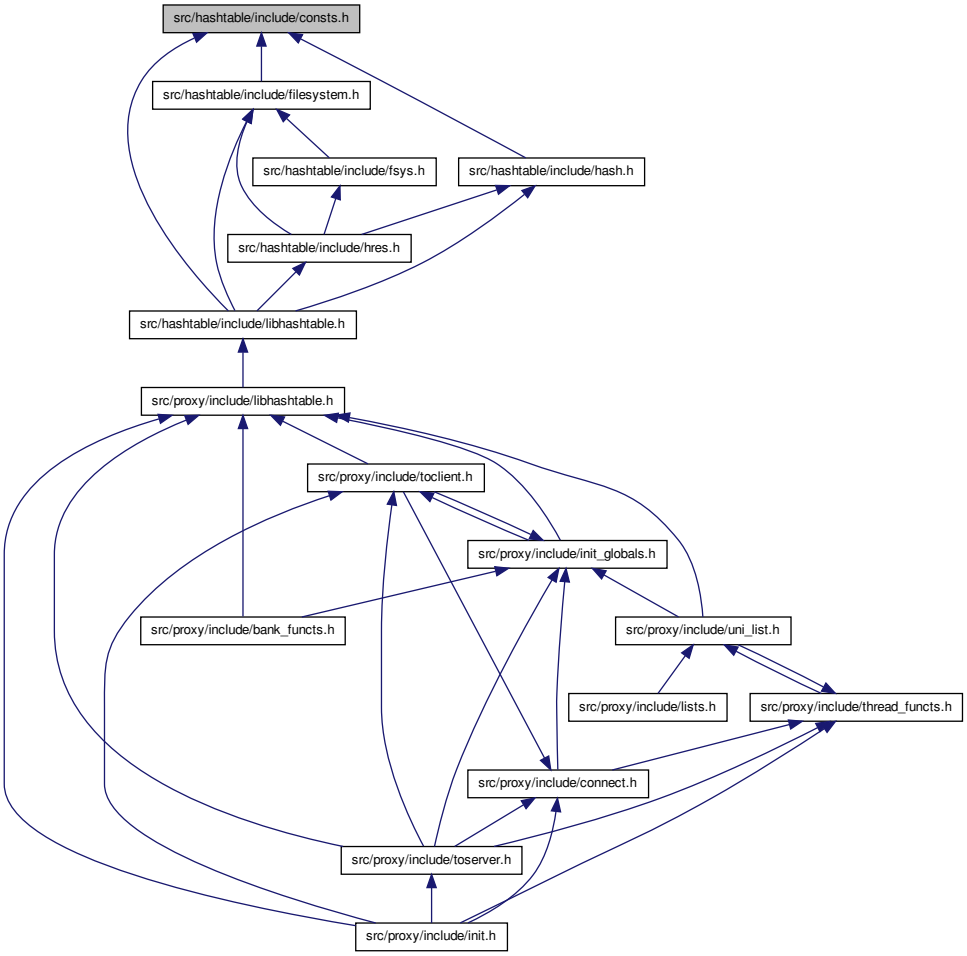
4.4.1.5 int map_cached_file (HRES * *hash,* char * *remote_file,* void ** *toret*)

4.4.1.6 int parseHead_time (char * *buf*)

parseHead_time() Semplificazione della funzione di parsing del file, per ottenerne solamente Il valore di expire

4.5 Riferimenti per il file src/hashtable/include/consts.h

Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define MAX_FILE 5001
- #define NHASH1 29989
- #define NHASH2 6661
- #define MOD 2000
- #define MULT1 31
- #define MULT2 47

- #define HASH_SIZE MOD
- #define MAX_HRES MOD

4.5.1 Documentazione delle definizioni

4.5.1.1 #define HASH_SIZE MOD

4.5.1.2 #define MAX_FILE 5001

4.5.1.3 #define MAX_HRES MOD

4.5.1.4 #define MOD 2000

4.5.1.5 #define MULT1 31

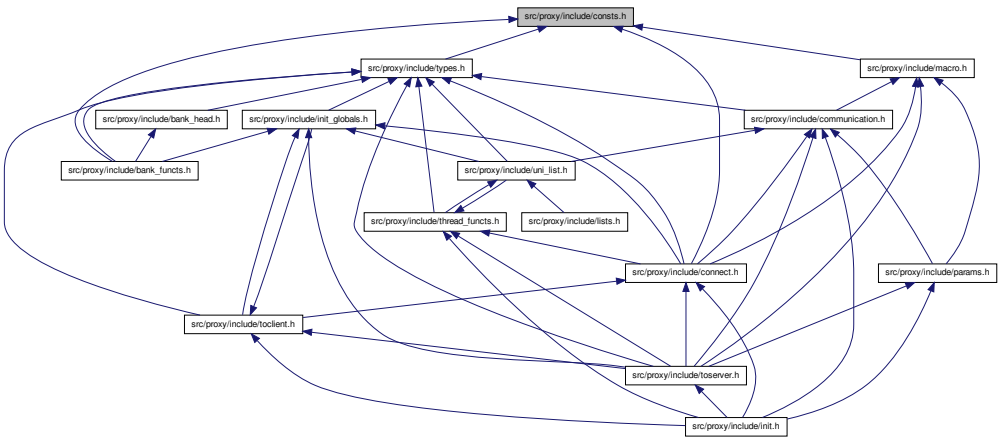
4.5.1.6 #define MULT2 47

4.5.1.7 #define NHASH1 29989

4.5.1.8 #define NHASH2 6661

4.6 Riferimenti per il file src/proxy/include/consts.h

Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define INTERVAL_NOT_FOUND 405

- #define NOT_FOUND 404
- #define WRONG_REQUEST 403
- #define UNKNOWN_ERROR 402
- #define INFO 202
- #define OK 200
- #define OK_RANGE 201
- #define OK_INFO 202
- #define MAXREF 10
- #define MAXIDREF 10
- #define MAXCONN 5
- #define MAX_uTIME 0
- #define MAX_TIME 3
- #define MAX_TIMESELECT 4
- #define PROXY_PORT 55554
- #define PROXY_IP "127.0.0.1"
- #define NO_MORE_THREAD (THREAD_COUNT==MAXCONN)
- #define LIMIT_FILE 5000
- #define MAX_PATH 2048
- #define MAX_FILE 5001
- #define MAX_BUFF 5001
- #define MAXREQUEST MAX_FILE
- #define GET 1
- #define INF 2
- #define UNKNOWN_ERROR_SIZE 6
- #define SOCKET_ERROR -1
- #define START_COLOR "\033["
- #define END_COLOR "m"
- #define RESET_COLOR "\033[0;37m"
- #define BOLD 1
- #define BLACK 30
- #define RED 31
- #define GREEN 32
- #define YELLOW 33
- #define BLUE 34
- #define MAGENTA 35
- #define CYAN 36
- #define WHITE 37

4.6.1 Documentazione delle definizioni

4.6.1.1 #define BLACK 30

4.6.1.2 #define BLUE 34

4.6.1.3 #define BOLD 1

4.6.1.4 #define CYAN 36

4.6.1.5 #define END_COLOR "m"

4.6.1.6 #define GET 1

4.6.1.7 #define GREEN 32

4.6.1.8 #define INF 2

4.6.1.9 #define INFO 202

4.6.1.10 #define INTERVAL_NOT_FOUND 405

4.6.1.11 #define LIMIT_FILE 5000

4.6.1.12 #define MAGENTA 35

4.6.1.13 #define MAX_BUFF 5001

4.6.1.14 #define MAX_FILE 5001

4.6.1.15 #define MAX_PATH 2048

4.6.1.16 #define MAX_TIME 3

4.6.1.17 #define MAX_TIMESELECT 4

4.6.1.18 #define MAX_uTIME 0

4.6.1.19 #define MAXCONN 5

4.6.1.20 #define MAXIDREF 10

4.6.1.21 #define MAXREF 10

4.6.1.22 #define MAXREQUEST MAX_FILE

4.6.1.23 #define NO_MORE_THREAD (THREAD_COUNT==MAXCONN)

4.6.1.24 #define NOT_FOUND 404

4.6.1.25 #define OK 200

4.6.1.26 #define OK_INFO 202

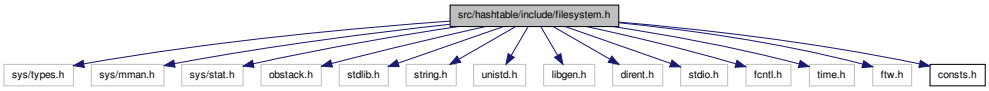
4.6.1.27 #define OK_RANGE 201

4.6.1.28 #define PROXY_IP "127.0.0.1"

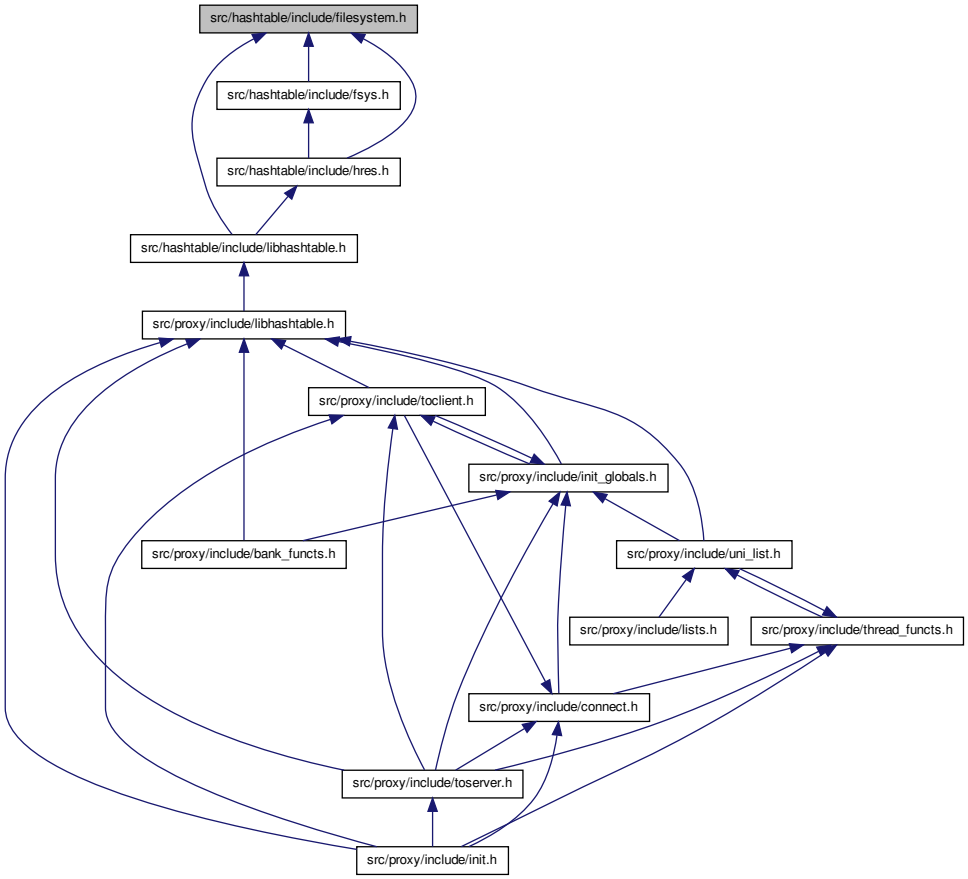
4.6.1.29 #define PROXY_PORT 55554

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <obstack.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <libgen.h>
#include <dirent.h>
#include <stdio.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>
#include "consts.h"
```

Grafo delle dipendenze di inclusione per filesystem.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define _GNU_SOURCE
- #define close_map(mem)
- #define cache_free() recursiveDelete("cache")
- #define FILE_SIZE sizeof(RES_FILE)
- #define NOW ((unsigned int)(time(NULL)))
- #define PAST(x) (NOW-((unsigned int)x))
- #define EXPIRED(creation, expiry) (((unsigned int)expiry)>PAST(creation))

Funzioni

- char * obtain_local (char *remote)

- void recursiveDelete (char ∗cache)
- char file_exists (char ∗filename, unsigned int ∗time, char ∗is_blank, struct stat ∗retbuf, int ∗pfd)
- char folder_empty (char ∗dirname)
- char ∗ resource_exists (char ∗remote, int deallocate, char ∗result, unsigned int ∗time, char ∗blank, struct stat ∗retbuf, int ∗pfd)
- void re_new_resource (char ∗cache_path, struct stat ∗retbuf, int ∗pfd)
- void resource_remove (char ∗path)

4.7.1 Documentazione delle definizioni

4.7.1.1 #define _GNU_SOURCE

4.7.1.2 #define cache_free( ) recursiveDelete("cache")

cache_free(): Funzione per cancellare dalla cache tutti i files e le cartelle contenute

4.7.1.3 #define close_map(*mem*)

Valore:

```
do {                                                \
                                int val = munmap((void*)mem,FILE_SIZE); \
                                kassert(val>=0,  "Error unmapping file");\
                        }while(0)
```

close_map(): Effettua la chiusura della mappatura del file

Parametri

mem,:	area di memoria da demappare

4.7.1.4 #define EXPIRED(*creation, expiry*) (((unsigned int)expiry)>PAST(creation))

4.7.1.5 #define FILE_SIZE sizeof(RES_FILE)

4.7.1.6 #define NOW ((unsigned int)(time(NULL)))

4.7.1.7 #define PAST(*x*) (NOW-((unsigned int)x))

4.7.2 Documentazione delle funzioni

4.7.2.1 char file_exists (char ∗ *filename,* unsigned int ∗ *time,* char ∗ *is_blank,* struct stat ∗ *retbuf,* int ∗ *pfd*)

Funzioni accessorie per la cache file_exist(): Verifica l'esistenza del file *filename*. Se il pathname esiste ed è un file, restituisce 1, se esso è una cartella restituisce 2, se non

esiste 0.

Parametri

filename,:	file locale del quale controllare l'esistenza
time,:	Conterrà il tempo da Epoch della creazione del file
is_blank,:	Restituisce, se passato come argomento, se il file è solamente troncato
retbuf,:	Restituisce lo fstat del file, se esiste.
pfd,:	Possibile (o NULL) puntatore al FileDescriptor: in quel caso il file non verrà chiuso

4.7.2.2 char folder_empty (char * *dirname*)

4.7.2.3 char* obtain_local (char * *remote*)

Funzioni fondamentali obtain_local(): Dato il pathning remoto, alloca in una stringa l'eventuale pathning locale corrispondente

Parametri

remote,:	percorso remoto del quale ottenere la conversione alla stringa corrente

4.7.2.4 void re_new_resource (char * *cache_path,* struct stat * *retbuf,* int * *pfd*)

re_new_resource(): Data il path della cache, crea la nuova risorsa: se le cartelle intermedie non esistono, esse vengono create, ed il file finale viene troncato ed aperto alla dimensione base massima. Restituisce la data di creazione del file

Parametri

cache_path,:	Identifica il percorso locale della risorsa
retbuf,:	Identifica lo stat del file che è stato aperto. Se non NULL viene aggiornato.
pfd,:	Identifica il FileDescriptor del nuovo file. Se non NULL viene aggiornato, altrimenti viene chiuso.

4.7.2.5 void recursiveDelete (char * *dirname*)

recursiveDelete(): Effettua l'eliminazione ricorsiva degli elementi all'interno della directory

Parametri

dirname,:	Percorso della cartella da svuotare ricorsivamente

4.7.2.6 char* resource_exists (char * *remote,* int *deallocate,* char * *result,* unsigned int * *time,* char * *blank,* struct stat * *retbuf,* int * *pfd*)

Funzioni fondamentali resource_exists(): Questa funzione verifica se il path remoto esista o meno. In base al valore del parametro *deallocate*, viene restituita o meno la stringa convertita della risorsa locale corrispondente

Parametri

remote,:	path remoto
deallocate,:	indica se deallocare l'indirizzo locale della risorsa corrispondente
result,:	verrà scritto in questo puntatore l'esistenza o meno del file
time,:	tempo di creazione della risorsa
blank,:	restituisce se il file è stato troncato o meno (e quindi vuoto)
retbuf,:	restituisce (se non NULL) il puntatore all'fstat del file in cache
pfd,:	Possibile (o NULL) puntatore al FileDescriptor: viene passato un fd aperto.

resource_exists(): Questa funzione verifica se il path remoto esista o meno. In base al valore del parametro *deallocate*, viene restituita o meno la stringa convertita della risorsa locale corrispondente

Parametri

remote,:	path remoto
deallocate,:	indica se deallocare l'indirizzo locale della risorsa corrispondente
result,:	verrà scritto in questo puntatore l'esistenza o meno del file
time,:	tempo di creazione della risorsa
blank,:	restituisce se il file è stato troncato o meno (e quindi vuoto)
retbuf,:	restituisce (se non NULL) il puntatore all'fstat del file in cache
pfd,:	Possibile (o NULL) puntatore al FileDescriptor: viene passato un fd aperto.

4.7.2.7 void resource_remove (char * *path*)

resource_remove(): Questa funzione effettua l'unlink sul file definito, ed inoltre via via eventualmente cancella le cartelle genitori se queste sono vuote.

Parametri

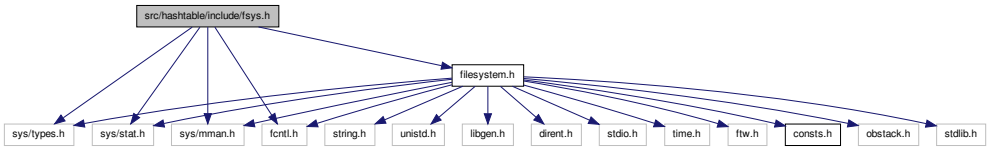
path,:	Risorsa remota

4.8 Riferimenti per il file src/hashtable/include/fsys.h

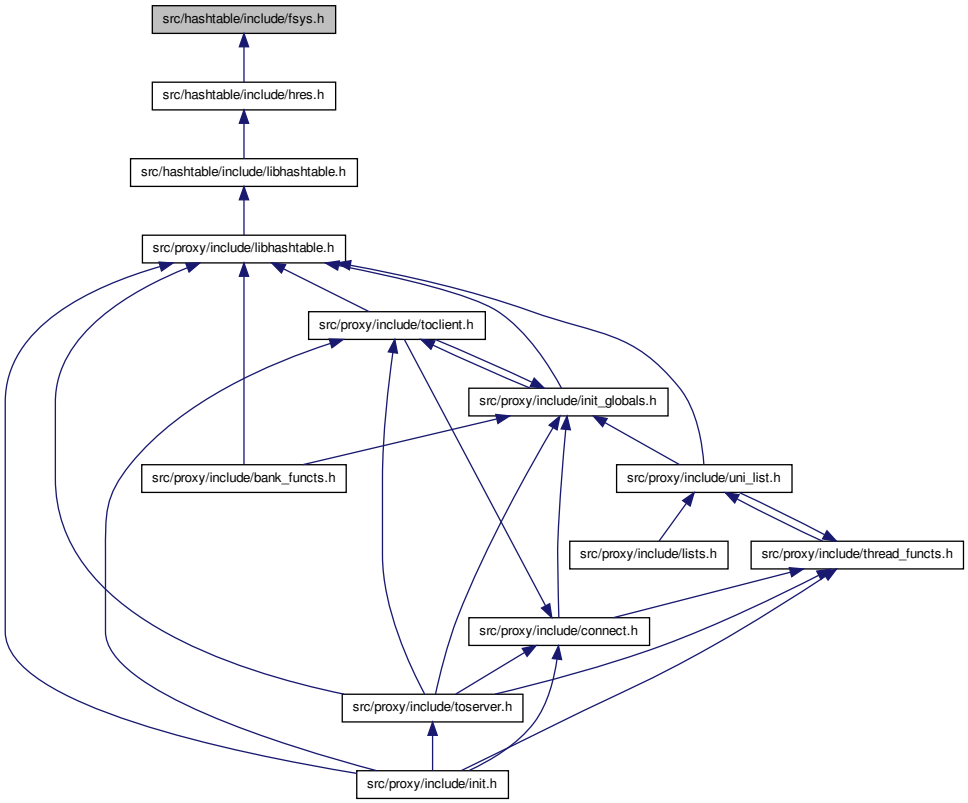
```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
```

```
#include "filesystem.h"
```

Grafo delle dipendenze di inclusione per fsys.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Funzioni

- int FileCreate (char *remote_resource, void *file)

- int FileOpen (char ∗remote_resource, void ∗∗update)
- int FileClose (void ∗update, int fd)

4.8.1 Documentazione delle funzioni

4.8.1.1 int FileClose (void ∗ *update,* int *fd*)

FileClose(): Effettua la chiusura del file mappato in memoria

Parametri

update,:	Puntatore alla risorsa mappata in memoria
fd,:	File descriptor della risorsa associata

Restituisce

Restituisce 0 in caso di insuccesso, altrimenti 1

4.8.1.2 int FileCreate (char ∗ *remote_resource,* void ∗ *file*)

FileCreate(): Questa funzione effettua la creazione del file, dato il nome della risorsa remota.

Parametri

remote_-resoruce,:	risorsa remota
expiry,:	tempo di expire
buffer,:	contenuto dell'informazione

Restituisce

restituisce 0 in caso di insuccesso, altrimenti 1

4.8.1.3 int FileOpen (char ∗ *remote_resource,* void ∗∗ *update*)

FileOpen(): Questa funzione effettua l'apertura di un file che si assume non essere scaduto.

Parametri

remote_-resource,:	indicazione della risorsa remota
update,:	Indicazione dell'area di allocazione del file in cache

Restituisce

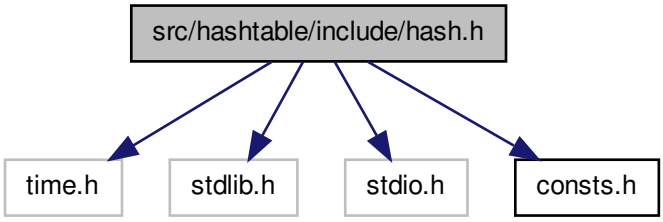
Restituisce 0 in caso di insuccesso, altrimenti il valore del filedescriptor

update = mmap(0,MAX_FILE,PROT_READ,MAP_SHARED,fd,0);

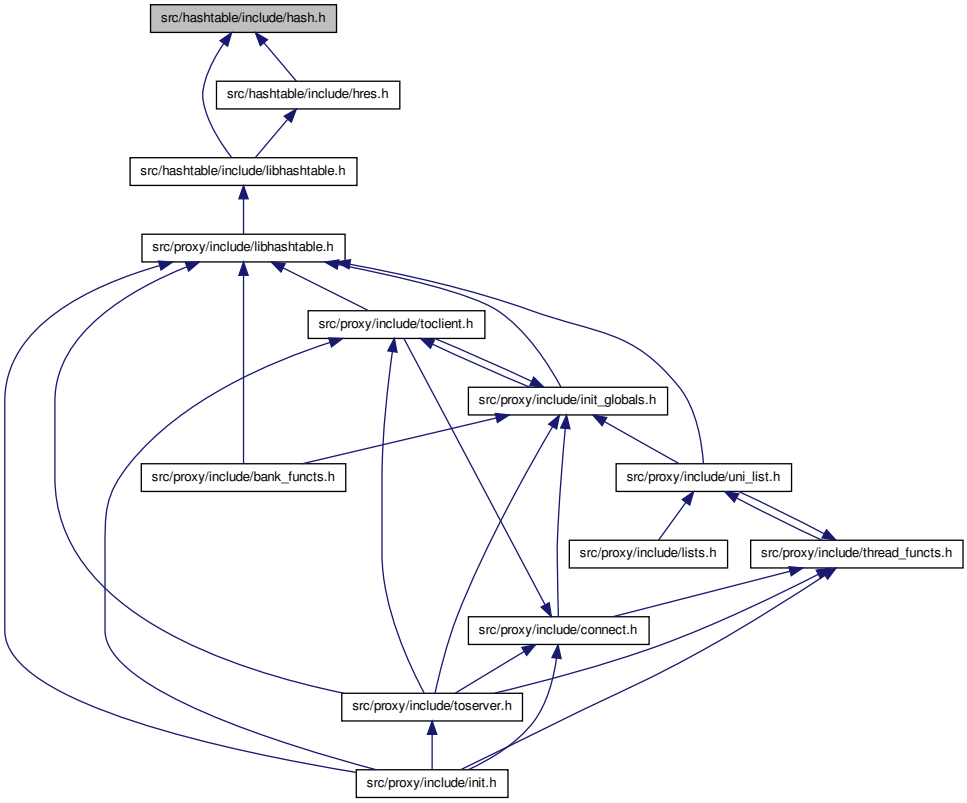
4.9 Riferimenti per il file src/hashtable/include/hash.h

```
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include "consts.h"
```

Grafo delle dipendenze di inclusione per hash.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Strutture dati

- struct hash_tbl

Definizioni

- #define _BSD_SOURCE

Funzioni

- void init_hashtbl (struct hash_tbl ∗p)
- unsigned int hashb (char ∗p, unsigned int MULT, unsigned int NHASH)
- unsigned int hash (char ∗p)
- int hash_exists (struct hash_tbl ∗tbl, char ∗p)
- void hash_insert (struct hash_tbl ∗tbl, char ∗p)

- void hash_remove (struct hash_tbl ∗tbl, char ∗p)
- int hash_occupied (struct hash_tbl ∗tbl)

4.9.1 Documentazione delle definizioni

4.9.1.1 #define _BSD_SOURCE

4.9.2 Documentazione delle funzioni

4.9.2.1 unsigned int hash (char ∗ *p*)

hash(): Funzione di hash

4.9.2.2 int hash_exists (struct **hash_tbl** ∗ *tbl,* char ∗ *p*)

hash_exists(): Verifica che una stringa sia presente all'interno della tavola

4.9.2.3 void hash_insert (struct **hash_tbl** ∗ *tbl,* char ∗ *p*)

hash_insert(): Effettua l'inserimento con collisioni

4.9.2.4 int hash_occupied (struct **hash_tbl** ∗ *tbl*)

hash_occupied(): Verifica se l'elemento è occupato

4.9.2.5 void hash_remove (struct **hash_tbl** ∗ *tbl,* char ∗ *p*)

hash_remove(): Effettua la rimozione di un elemento o di una collisione

4.9.2.6 unsigned int hashb (char ∗ *p,* unsigned int *MULT,* unsigned int *NHASH*)

hashb(): Funzione di base per lo hashing

4.9.2.7 void init_hashtbl (struct **hash_tbl** ∗ *p*)

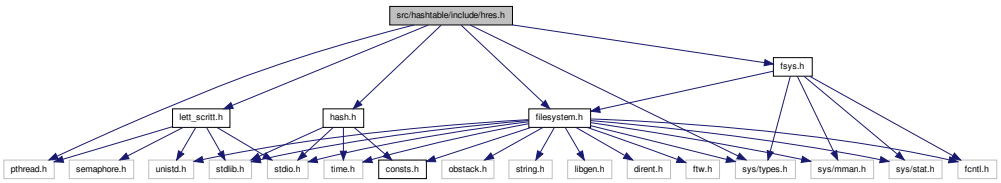
init_hashtbl(): Inizializzazione della hashtable

4.10 Riferimenti per il file src/hashtable/include/hres.h

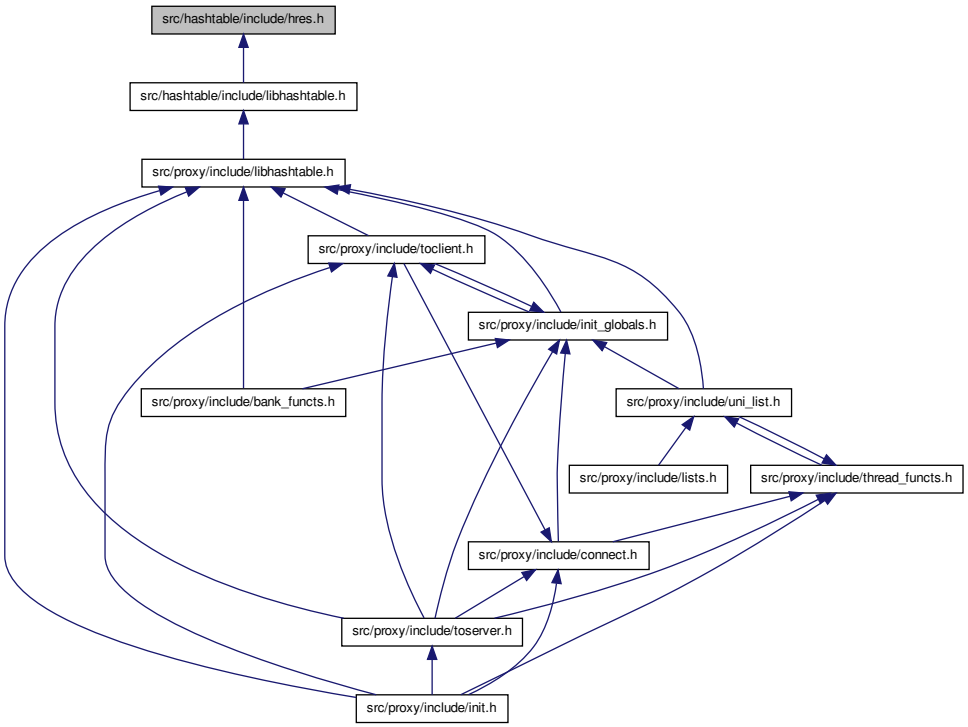
```
#include <sys/types.h>
#include <pthread.h>
#include "lett_scritt.h"
```

```
#include "filesystem.h"
#include "fsys.h"
#include "hash.h"
```

Grafo delle dipendenze di inclusione per hres.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Strutture dati

- struct HRES

Funzioni

- void init_hres (HRES *ptr)
- int is_inhash (HRES *phash, char *remote_file)
- int handle_file_filesystem (HRES *hash, char *remote_file, void *res, int creat_oth_delete)
- int map_cached_file (HRES *hash, char *remote_file, void **toret)
- void close_cached_file (HRES *hash, char *remote_file, void *toclose, int fd)

4.10.1 Documentazione delle funzioni

4.10.1.1 void close_cached_file (HRES * *hash,* char * *remote_file,* void * *toclose,* int *fd*)

4.10.1.2 int handle_file_filesystem (HRES * *phash,* char * *remote_file,* void * *res,* int *creat_oth_delete*)

handle_file_filesystem(): Funzione per la gestione controllata degli elementi del file-system

Parametri

hash,:	Struttura dati di hashing
remote_file,:	Indicazione del file remoto di risorsa
res,:	Indica la risorsa contentente le informazioni, non utilizzata per l'eliminazione del file
creat_oth_-delete,:	Indica se il valore è maggiore di zero, di creare il file, altrimenti di cancellarlo.

Restituisce

La funzione restituisce comunque -1 in caso di cancellazione, la posizione nella htable (0 based) in caso di creazione con successo, altrimenti -1

4.10.1.3 void init_hres (HRES * *ptr*)

init_hres(): Inizializzazione della struttura dati

4.10.1.4 int is_inhash (HRES * *phash,* char * *remote_file*)

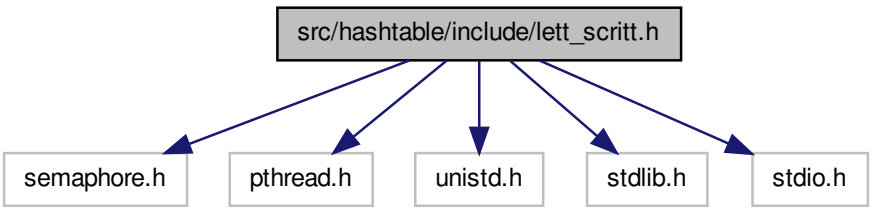
is_inhash(): Controlla se il file è presente in cache, e se esiste effettivamente all'interno del filesystem

4.10.1.5 int map_cached_file (HRES * *hash*, char * *remote_file*, void ** *toret*)

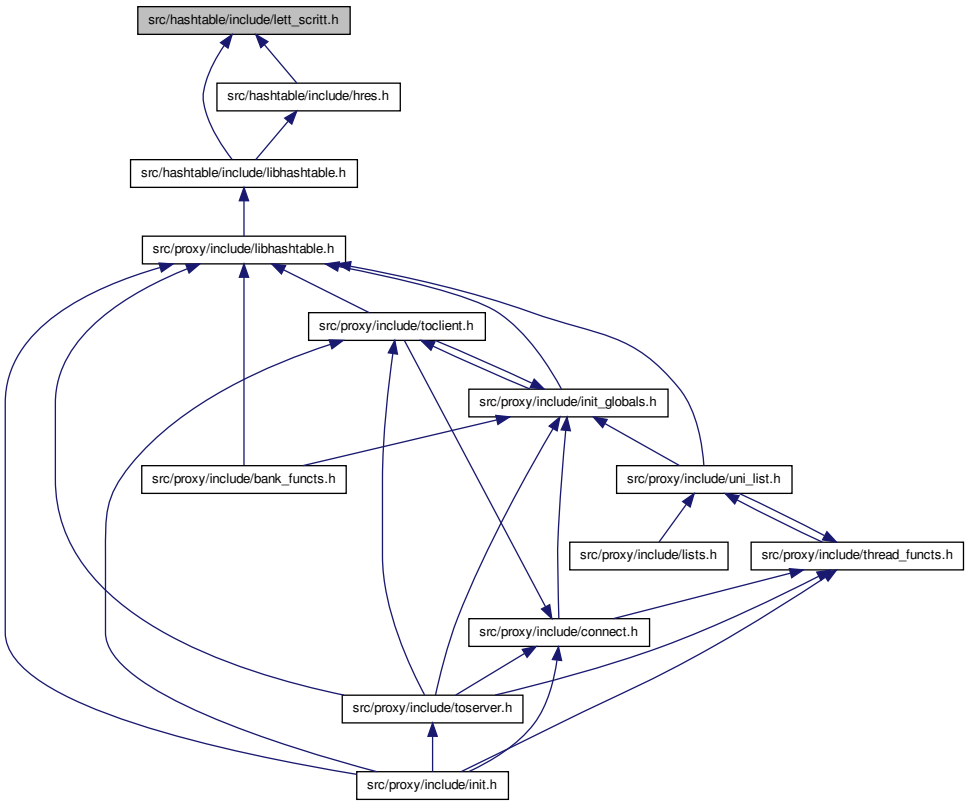
4.11 Riferimenti per il file src/hashtable/include/lett_scritt.h

```
#include <semaphore.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
```

Grafo delle dipendenze di inclusione per lett_scritt.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Strutture dati

- struct CS_struct

Definizioni

- #define WREAD 0
- #define WWRITE 1
- #define P(x) sem_wait(x)
- #define V(x) sem_post(x)

Ridefinizioni di tipo (typedef)

- typedef unsigned int ANYTYPE

Funzioni

- void init_struct (struct CS_struct *data)
- void struct_do_read (struct CS_struct *data)
- void struct_end_read (struct CS_struct *data)
- void struct_do_write (struct CS_struct *data)
- void struct_end_write (struct CS_struct *data)

4.11.1 Documentazione delle definizioni

4.11.1.1 #define P(*x*) sem_wait(x)

4.11.1.2 #define V(*x*) sem_post(x)

4.11.1.3 #define WREAD 0

4.11.1.4 #define WWRITE 1

4.11.2 Documentazione delle ridefinizioni di tipo (typedef)

4.11.2.1 typedef unsigned int ANYTYPE

4.11.3 Documentazione delle funzioni

4.11.3.1 void init_struct (struct CS_struct * *data*)

init_struct(): Inizializza la struttura dati passata per argomento, settando il numero dei lettori e degli scrittori, ed inizializzando il semaforo associato al dato

4.11.3.2 void struct_do_read (struct CS_struct * *data*)

struct_do_read(): Funzione di accesso mutuamente esclusivo in lettura, al termine della quale si può ottenere il dato richiesto.

Parametri

data,:	Indica la struttura dati sulla quale si effettua il lock
external,:	Indica un eventuale locking esterno da rilasciare e di cui riappropriarsi in seguito

4.11.3.3 void struct_do_write (struct CS_struct * *data*)

struct_do_write(): Funzione di accesso in scrittura in mutua esclusione, al termine della quale si può scrivere il dato richiesto

4.11.3.4 void struct_end_read (struct CS_struct * *data*)

struct_end_read(): Funzione di rilascio della mutua esclusione in lettura sulla struttura dati

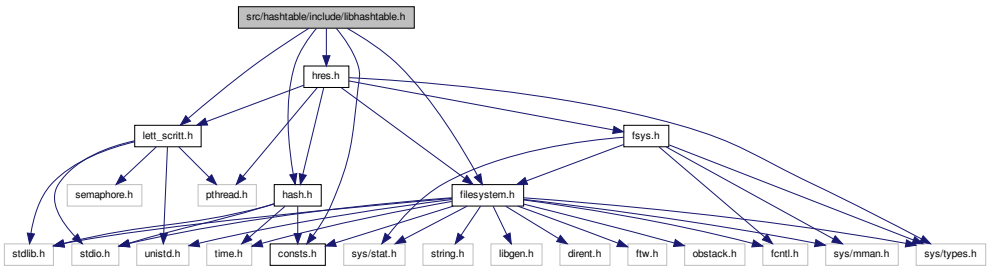
4.11.3.5 void struct_end_write (struct CS_struct * *data*)

struct_end_write(): Funzione di rilascio della mutua esclusione in scrittura sulla struttura dati

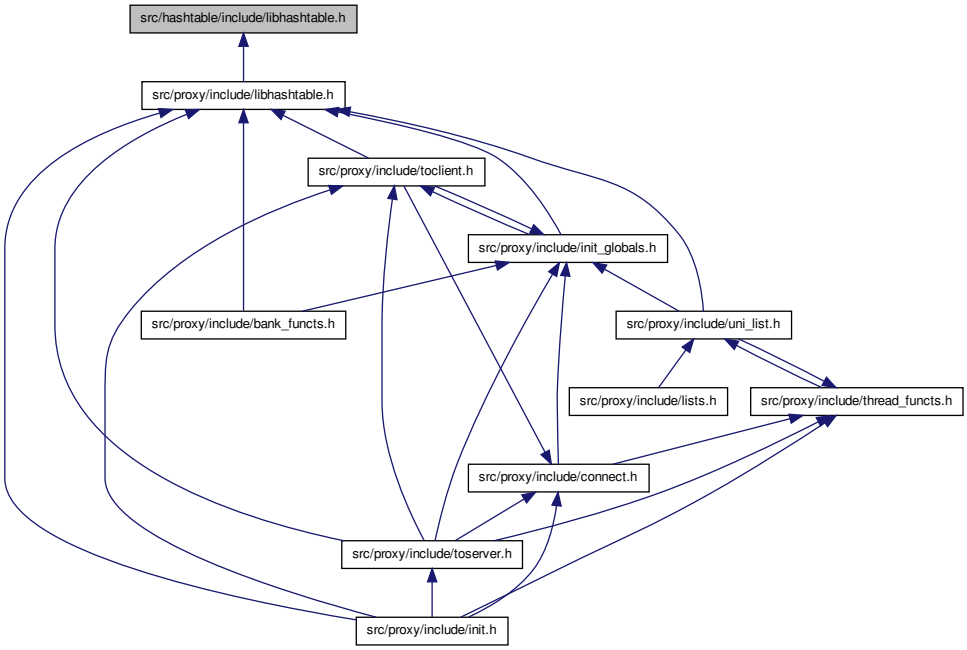
4.12 Riferimenti per il file src/hashtable/include/libhashtable.h

```
#include "lett_scritt.h"
#include "filesystem.h"
#include "consts.h"
#include "hash.h"
#include "hres.h"
```

Grafo delle dipendenze di inclusione per libhashtable.h:



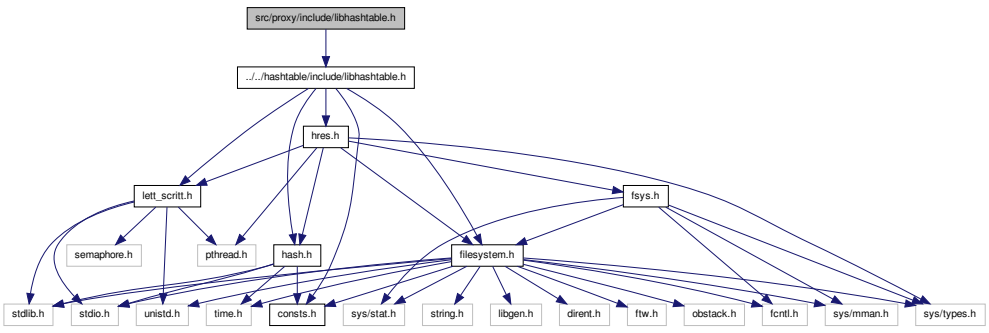
Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



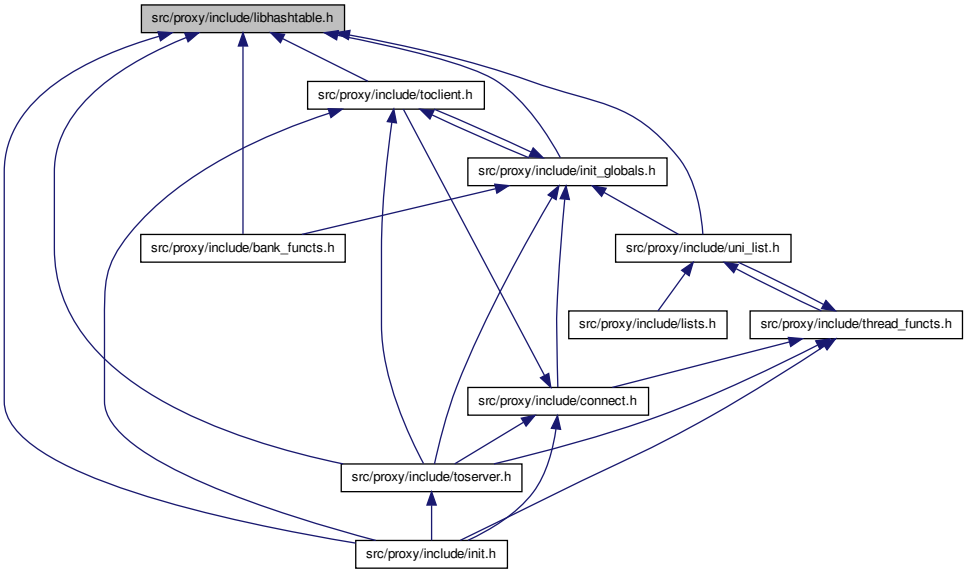
4.13 Riferimenti per il file src/proxy/include/libhashtable.h

```
#include "../../hashtable/include/libhashtable.h"
```

Grafo delle dipendenze di inclusione per libhashtable.h:



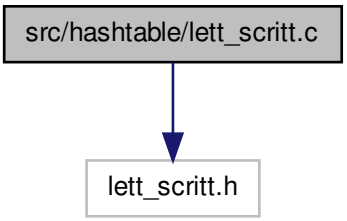
Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



4.14 Riferimenti per il file src/hashtable/lett_scritt.c

```
#include "lett_scritt.h"
```

Grafo delle dipendenze di inclusione per lett_scritt.c:



Funzioni

- void init_struct (struct CS_struct *data)
- void struct_do_read (struct CS_struct *data)
- void struct_end_read (struct CS_struct *data)
- void struct_do_write (struct CS_struct *data)
- void struct_end_write (struct CS_struct *data)

4.14.1 Documentazione delle funzioni

4.14.1.1 void init_struct (struct CS_struct * *data*)

init_struct(): Inizializza la struttura dati passata per argomento, settando il numero dei lettori e degli scrittori, ed inizializzando il semaforo associato al dato

4.14.1.2 void struct_do_read (struct CS_struct * *data*)

struct_do_read(): Funzione di accesso mutuamente esclusivo in lettura, al termine della quale si può ottenere il dato richiesto.

Parametri

data,:	Indica la struttura dati sulla quale si effettua il lock
external,:	Indica un eventuale locking esterno da rilasciare e di cui riappropriarsi in seguito

4.14.1.3 void struct_do_write (struct CS_struct * *data*)

struct_do_write(): Funzione di accesso in scrittura in mutua esclusione, al termine della quale si può scrivere il dato richiesto

4.14.1.4 void struct_end_read (struct CS_struct * *data*)

struct_end_read(): Funzione di rilascio della mutua esclusione in lettura sulla struttura dati

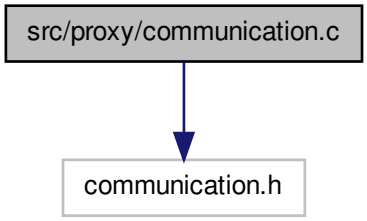
4.14.1.5 void struct_end_write (struct CS_struct * *data*)

struct_end_write(): Funzione di rilascio della mutua esclusione in scrittura sulla struttura dati

4.15 Riferimenti per il file src/proxy/communication.c

```
#include "communication.h"
```

Grafo delle dipendenze di inclusione per communication.c:



Funzioni

- int haspos (char ∗text, char ∗BASE, int pos)
- char ∗ surfto (char ∗text, char base, int text_size)
- char ∗ atreturns (char ∗buf, int len)
- static char ∗ atreturn (char ∗buf, int len)
- int dorequest (char ∗request, int TYPE, char ∗path, int low, int hight)
- int doreply (char ∗reply, int TYPE, int len, int empire, int low, int hight, char ∗content)
- int parseHead (char ∗buf, int buflen, int ∗len, unsigned int ∗expire, int ∗low, int ∗hight, char ∗∗cont)
- request ∗ parseRequest (char ∗buf, int len)
- char ∗ getIP (char ∗URL, char ∗dest)
- char ∗ getPort (char ∗URLp, char ∗dest)
- char ∗ getfirstlevel_next (char ∗text, char ∗buf, int len, int text_len)
- char ∗ get_resource_alloc (char ∗text, char ∗∗next)

Variabili

- int oldoutptfd

4.15.1 Documentazione delle funzioni

4.15.1.1 static char∗ atreturn (char ∗ *buf*, int *len*) [static]

atreturn()

FUNZIONE DI PARSING/LEXING ---------- Restituisce eventualmente il puntatore alla stringa dopo una andata a capo

4.15.1.2 char* atreturns (char * *buf*, int *len*)

atreturns()

FUNZIONE DI PARSING/LEXING ---------- Restituisce eventualmente il puntatore alla stringa dopo due andate a capo

4.15.1.3 int doreply (char * *reply*, int *TYPE*, int *len*, int *empire*, int *low*, int *hight*, char * *content*)

doreply() Funzione che stampa all'interno di un buffer di caratteri la risposta da inoltrare. Ritorna 0 se TYPE è un valore compreso tra le costanti predefinite.

Parametri

reply,:	Buffer dove viene stampato il risultato
TYPE,:	Tipologia della richeista da inoltrare (const.h)
len,:	Lunghezza della risposta che si sta ottenendo con la funzione
empire,:	Expiry time
low,:	Eventuale low range
hight,:	Eventuale hight range. Se hight<low, allora non viene inserito l'upper
content,:	Eventuale contenuto del messaggio

4.15.1.4 int dorequest (char * *request*, int *TYPE*, char * *path*, int *low*, int *hight*)

dorequest() Funzione che stampa all'interno di un buffer di caratteri la richiesta da inoltrare Restituisce 0 se la generazione è andata a buon fine, altrimenti 1 (se TYPE non è un valore previsto da const.h)

Parametri

request,:	Percorso dove salvare il valore della richiesta
TYPE,:	Tipo della richiesta da generare
path,:	Percorso remoto della risorsa.
low,:	Low Range della risorsa ottenuta
hight,:	Hight Range della risorsa ottenuta. Se il valore è inferiore a low, allora inserisce solo il low.

4.15.1.5 char* get_resource_alloc (char * *text*, char ** *next*)

get_resource_alloc(): Restituisce una stringa ove è allocata la risorsa indicato dal buffer passato come argomento

Parametri

text,:	puntatore al buffer di memoria dove è contenuta la risposta
next,:	puntatore ad un puntatore a carattere che verrà aggiornato con la posizione dalla quale riprendere la scansione con la get_resource_alloc

4.15.1.6 char* getfirstlevel_next (char * *text,* char * *buf,* int *len,* int *text_len*)

getfirstlevel_next() Questa funzione, dato un campo di test *text* ed un buffer *buf* di dimensioni *len*, se trova un qualsiasi tipo di risorsa memorizza il valore ottenuto all'interno del suddetto buffer, altrimenti restituisce NULL

Parametri

text,:	Buffer di input
buf,:	BUffer di output
len,:	Lunghezza del buffer di output
text_len,:	Lunghezza del buffer di input

4.15.1.7 char* getIP (char * *URL,* char * *dest*)

getIP() Questa funzione restituisce in dest il valore dell'IP fornito: restituisce NULL se il matching è stato scorretto, altrimenti il punto successivo da cui riprendere l'analisi della stringa

Parametri

URL,:	Contiene l'URL partendo da "mhttp://"
dest,:	Bufffer di destinazione dove salvare la stringa remota

4.15.1.8 char* getPort (char * *URLp,* char * *dest*)

getPort() Questa funzione ottiene la porta del nostro URL, concordemente all'analisi effettuata precedentemente per trovare l'IP. Si comporta in modo analogo a getIP(): il puntatore restituito in caso di procedura corretta è il pathening

4.15.1.9 int haspos (char * *text,* char * *BASE,* int *pos*)

haspos() Dato un campo di testo *text* dove deve essere cercato *BASE* dalla posizione *pos* in *text*, restituisce -1 se BASE è inesistente dal primo suo carattere, altrimenti la posizione successiva all'ultimo carattere in BASE.

4.15.1.10 int parseHead (char * *buf,* int *buflen,* int * *len,* unsigned int * *expire,* int * *low,* int * *hight,* char ** *cont*)

parseHead() Effettua il parsing della stringa all'interno di buf. Viene restituito il numero della risposta se è una risposta ben formata, altrimenti 0 (anche se buf è NULL). Vengono passti per parametri i valori assunti dallo header del messaggio.

Parametri

buf,:	Eventuale buffer dal quale leggere la risposta ricevuta
buflen,:	Lunghezza della parte di memoria effettiva in memoria

len,:	Lunghezza del parametro LEN della richiesta INF
expire,:	Expire time fornito dal messaggio dalla richiesta inf
low,:	Low range
hight,:	Hight Range
cont,:	Punto dal quale continuare la lettura (puntatore al body)

4.15.1.11 request* parseRequest (char * *buf,* int *len*)

parseRequest() Funzione che effettua il parsing della richiesta remota.

Parametri

buf,:	Buffer contenente la risorsa
len,:	Lunghezza della risorsa del buffer

4.15.1.12 char* surfto (char * *text,* char *base,* int *text_size*)

surfto() Dato un campo di testo *text*, si vuole cercare dal suo inizio il carattere base: verrà restituita la posizione successiva al ritrovamento del carattere, altrimenti NULL

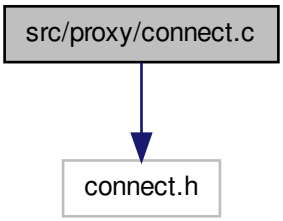
4.15.2 Documentazione delle variabili

4.15.2.1 int oldoutptfd

4.16 Riferimenti per il file src/proxy/connect.c

```
#include "connect.h"
```

Grafo delle dipendenze di inclusione per connect.c:



Definizioni

- #define INIT_CONNECT(x) memset((void∗)(x),0,sizeof(struct connection))

Funzioni

- void fallisci_tutto (struct connection ∗ptr)
- int establishConnection (struct connection ∗ptr)
- int init_connect_struct (struct connection ∗local, char ∗resourcepath, int client)
- int server_base_send (int fd, char ∗buf, int len)
- int server_complete_send (struct connection ∗ptr, char ∗buf, int len)
- int send_request (struct connection ∗ptr)
- int base_read (int fd, char ∗buf, int len)
- int server_complete_read (struct connection ∗ptr, char ∗buf, int len)
- void invia_client_termina (struct connection ∗local, char ∗res)
- int session (char ∗obtained_remote, char ∗resourcepath, int client)

4.16.1 Documentazione delle definizioni

4.16.1.1 #define INIT_CONNECT(*x*) memset((void∗)(x),0,sizeof(struct connection))

4.16.2 Documentazione delle funzioni

4.16.2.1 int base_read (int *fd,* char ∗ *buf,* int *len*)

base_read(): Funzione base di lettura

Parametri

fd,:	Socket di comunicazione dal quale effettuare la lettura
buf,:	Buffer all'interno del quale effettuare la scrittura
len,:	Lunghezza della risorsa da scaricare

4.16.2.2 int establishConnection (struct connection ∗ *ptr*)

establishConnection(): Inizia la connessione con il server, solamente se ciò non è stato effettuato preventivamente

Parametri

ptr,:	Puntatore allo stato della connessione con il server

4.16.2.3 void fallisci_tutto (struct connection ∗ *ptr*)

fallisci_tutto(): Chiude definitivamente la connessione, ed eventualmente effettua l'inoltro della richiesta tramite lettura in cache.

Parametri

ptr,:	Parametro della connessione con il server

4.16.2.4 int init_connect_struct (struct connection * *local,* char * *resourcepath,* int *client*)

init_connect_struct(): Effettua l'inizializzazione della struttura dati della connessione con il server

Parametri

local,:	Puntatore alla struttura dati di stato di connessione con il server
resourcepath,:	Stringa completa della risorsa da chiedere al server
client,:	Socket di comunicazione verso client

4.16.2.5 void invia_client_termina (struct connection * *local,* char * *res*)

invia_client_termina(): Effettua l'invio al client della risorsa ottenuta

Parametri

local,:	Stato di connessione con il server
res,:	Risorsa da inviare al client

4.16.2.6 int send_request (struct connection * *ptr*)

send_request(): Effettua l'invio completo con il server: se questo dovesse fallire, viene inoltre chiusa la comunicazione con il client

Parametri

ptr,:	Puntatore allo stato di connessione con il server

4.16.2.7 int server_base_send (int *fd,* char * *buf,* int *len*)

server_base_send(): Effettua l'invio di una richiesta lato server

Parametri

fd,:	File descriptor per l'invio
buf,:	Buffer contenente l'informazione da inviare
len,:	Lunghezza della risorsa da inviare

4.16.2.8 int server_complete_read (struct connection * *ptr,* char * *buf,* int *len*)

server_complete_read(): Effettua una lettura completa con gestione dei tentativi verso il server.

Parametri

ptr,:	Puntatore allo stato di connessione con il server
buf,:	Buffer nel quale memorizzare la risposta
len,:	Lunghezza dell'informazione da leggere

4.16.2.9 int server_complete_send (struct connection * *ptr,* char * *buf,* int *len*)

server_complete_send(): Effettua un invio completo al server, contemplando la gestione dei tentativi.

Parametri

ptr,:	Struttura dati riguardante lo stato di connessione con il server

4.16.2.10 int session (char * *obtained_remote,* char * *resourcepath,* int *client*)

session(): Instaura una sessione completa con il server, comprensiva della gestione dei tentativi

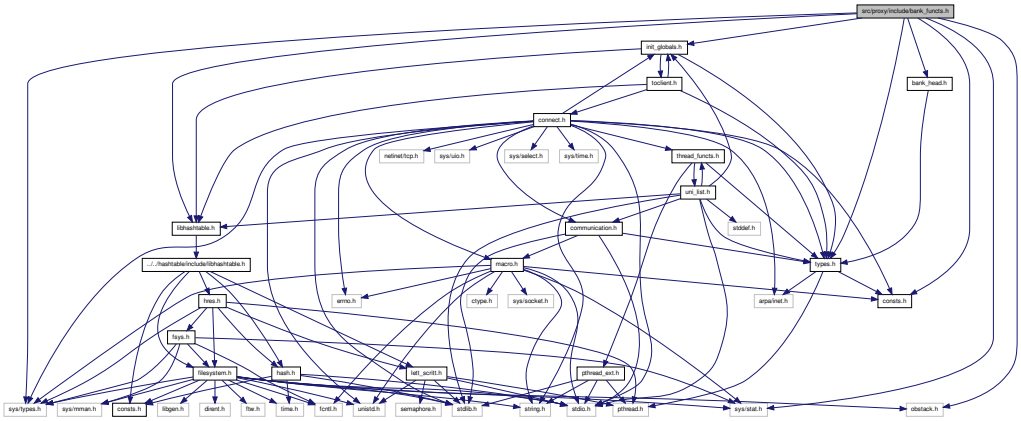
Parametri

obtained_remote,:	Buffer all'interno del quale verrà memorizzata la risorsa
resourcepath,:	Percorso remoto della risorsa da ottenere
client,:	Socket del client

4.17 Riferimenti per il file src/proxy/include/bank_functs.h

```
#include <sys/types.h>
#include <sys/stat.h>
#include <obstack.h>
#include "libhashtable.h"
#include "bank_head.h"
#include "consts.h"
#include "types.h"
#include "init_globals.h"
```

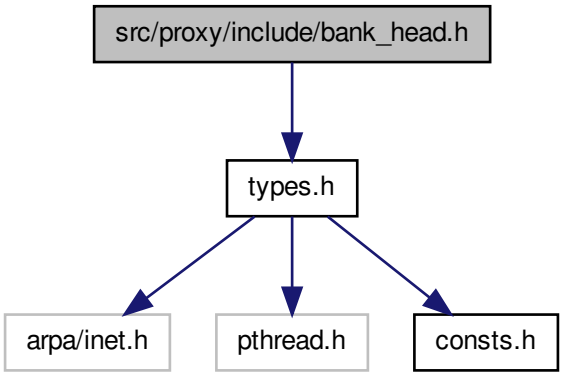
Grafo delle dipendenze di inclusione per bank_functs.h:



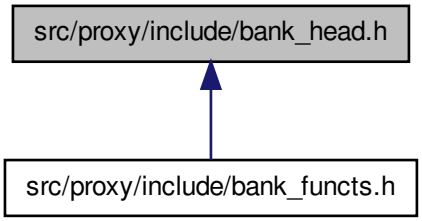
4.18 Riferimenti per il file src/proxy/include/bank_head.h

```
#include "types.h"
```

Grafo delle dipendenze di inclusione per bank_head.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Funzioni

- char cache_exists (char ∗remote_path, int dealloca, char ∗∗local_path, char flag)
- RES ∗ cache_init (RES ∗Threads_Resource, char existance_val, char ∗remote_-path)
- int cache_init_socket (sockets ∗Bart, int max_socks, int no, int remote_file_size)
- void cache_done (RES ∗resource)

4.18.1 Documentazione delle funzioni

4.18.1.1 void cache_done (RES ∗ *resource*)

cache_done(): Questa funzione viene chiamata per effettuare l'unmapping della memoria. Deve essere effettuata garantendo precedentemente la mutua esclusione con la mutex_could_close --RES--.

Parametri

resource,:	risorsa che deve essere chiusa
do_free_-not_unmap,:	identifica se, precedentemente, il file era stato troncato

4.18.1.2 char cache_exists (char ∗ *remote_path,* int *dealloca,* char ∗∗ *local_path,* char *flag*)

cache_exists(): Dato il pathening remoto, controlla che in cache esista la risorsa corrispondente, o che la risorsa non sia scaduta. Se scade, allora la risorsa viene cancellata e restituito un valore falso.

Parametri

remote_path,:	Percorso della risorsa remota da richiedere al server
dealloca,:	Se è settato a vero, non viene settato il valore local_path, altrimenti viene aggiornato il suo valore
local_path,:	Doppio puntatore a carattere: se passato per riferimento un puntatore a carattere, aggiorna il suo valore con la stringa remota; non viene modificato il valore se *dealloca* è settato a vero.

Restituisce

: Restituisce 0 se non esiste, 1 se esiste e 2 se esiste ma è troncato

4.18.1.3 RES* cache_init (RES * *Threads_Resource,* char *existance_val,* char * *remote_path*)

cache_init(): Funzione per la inizializzazione della struttura della risorsa per la cache.

Parametri

Threads_Resource,:	Risorsa locale associata al thread corrente. Può essere settata a NULL se existance_val == 2
exsistance_val,:	Valore di esistenza associato al file corrente (ottenuto da cache_exists)
remote_path,:	Nome della risorsa remota

Restituisce

: Restituisce o lo stesso puntatore alla risorsa che è stata associata come parametro.

4.18.1.4 int cache_init_socket (sockets * *Bart,* int *max_socks,* int *no,* int *remote_file_size*)

cache_init_socket(): Data la struttura dati RES come argomento e data la dimensione del file, effettuo la divisione nell'acquisizone della risorsa. Ritorna 0 in caso di successo.

Parametri

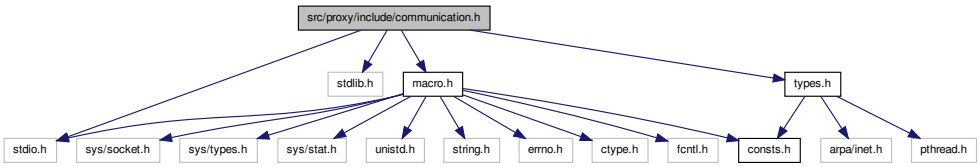
Bart,:	Struttura dati da aggiornare
max_socks,:	Numero totale dei sockets utilizzati dal thread corrente
no,:	Numero del socket corrente, contando da 1
remote_file_size,:	Dimensione del file remoto

4.19 Riferimenti per il file src/proxy/include/communication.h

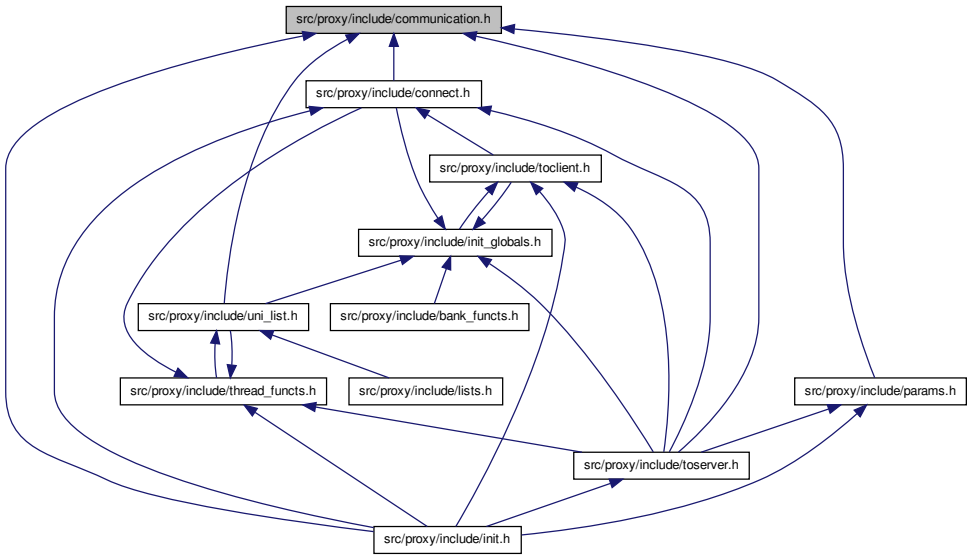
```
#include <stdio.h>
```

```
#include <stdlib.h>
#include "macro.h"
#include "types.h"
```

Grafo delle dipendenze di inclusione per communication.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define _GNU_SOURCE
- #define distance(ptr1, ptr2) ((!(sizeof(ptr1)==sizeof(ptr2))) ? -1 : (((ptr2)-(ptr1))/sizeof(*ptr1)))

Funzioni

- char * get_resource_alloc (char *text, char **next)
- int dorequest (char *request, int TYPE, char *path, int low, int hight)
- int doreply (char *reply, int TYPE, int len, int empire, int low, int hight, char *content)
- request * parseRequest (char *buf, int len)
- int parseHead (char *buf, int buflen, int *len, unsigned int *expire, int *low, int *hight, char **cont)
- char * getIP (char *URL, char *dest)
- char * getPort (char *URLp, char *dest)

4.19.1 Documentazione delle definizioni

4.19.1.1 #define _GNU_SOURCE

4.19.1.2 #define distance(*ptr1, ptr2*) ((!(sizeof(ptr1)==sizeof(ptr2))) ? -1 : (((ptr2)-(ptr1))/sizeof(*ptr1)))

distance()

MACRO DI CALCOLO ---------- Permette di effettuare il calcolo in quale posizione si trova ptr2 rispetto a ptr1, sapendo che sono puntatori con la stessa dimensione in memoria

4.19.2 Documentazione delle funzioni

4.19.2.1 int doreply (char * *reply,* int *TYPE,* int *len,* int *empire,* int *low,* int *hight,* char * *content*)

doreply() Funzione che stampa all'interno di un buffer di caratteri la risposta da inoltrare. Ritorna 0 se TYPE è un valore compreso tra le costanti predefinite.

Parametri

reply,:	Buffer dove viene stampato il risultato
TYPE,:	Tipologia della richeista da inoltrare (const.h)
len,:	Lunghezza della risposta che si sta ottenendo con la funzione
empire,:	Expiry time
low,:	Eventuale low range
hight,:	Eventuale hight range. Se hight<low, allora non viene inserito l'upper
content,:	Eventuale contenuto del messaggio

4.19.2.2 int dorequest (char * *request,* int *TYPE,* char * *path,* int *low,* int *hight*)

dorequest() Funzione che stampa all'interno di un buffer di caratteri la richiesta da inoltrare Restituisce 0 se la generazione è andata a buon fine, altrimenti 1 (se TYPE

non è un valore previsto da const.h)

Parametri

request,:	Percorso dove salvare il valore della richiesta
TYPE,:	Tipo della richiesta da generare
path,:	Percorso remoto della risorsa.
low,:	Low Range della risorsa ottenuta
hight,:	Hight Range della risorsa ottenuta. Se il valore è inferiore a low, allora inserisce solo il low.

4.19.2.3 char* get_resource_alloc (char * *text,* char ** *next*)

get_resource_alloc(): Restituisce una stringa ove è allocata la risorsa indicato dal buffer passato come argomento

Parametri

text,:	puntatore al buffer di memoria dove è contenuta la risposta
next,:	puntatore ad un puntatore a carattere che verrà aggiornato con la posizione dalla quale riprendere la scansione con la get_resource_alloc

4.19.2.4 char* getIP (char * *URL,* char * *dest*)

getIP() Questa funzione restituisce in dest il valore dell'IP fornito: restituisce NULL se il matching è stato scorretto, altrimenti il punto successivo da cui riprendere l'analisi della stringa

Parametri

URL,:	Contiene l'URL partendo da "mhttp://"
dest,:	Bufffer di destinazione dove salvare la stringa remota

4.19.2.5 char* getPort (char * *URLp,* char * *dest*)

getPort() Questa funzione ottiene la porta del nostro URL, concordemente all'analisi effettuata precedentemente per trovare l'IP. Si comporta in modo analogo a getIP(): il puntatore restituito in caso di procedura corretta è il pathening

4.19.2.6 int parseHead (char * *buf,* int *buflen,* int * *len,* unsigned int * *expire,* int * *low,* int * *hight,* char ** *cont*)

parseHead() Effettua il parsing della stringa all'interno di buf. Viene restituito il numero della risposta se è una risposta ben formata, altrimenti 0 (anche se buf è NULL). Vengono passti per parametri i valori assunti dallo header del messaggio.

Parametri

buf,:	Eventuale buffer dal quale leggere la risposta ricevuta
buflen,:	Lunghezza della parte di memoria effettiva in memoria
len,:	Lunghezza del parametro LEN della richiesta INF
expire,:	Expire time fornito dal messaggio dalla richiesta inf
low,:	Low range
hight,:	Hight Range
cont,:	Punto dal quale continuare la lettura (puntatore al body)

4.19.2.7 request* parseRequest (char * *buf,* int *len*)

parseRequest() Funzione che effettua il parsing della richiesta remota.

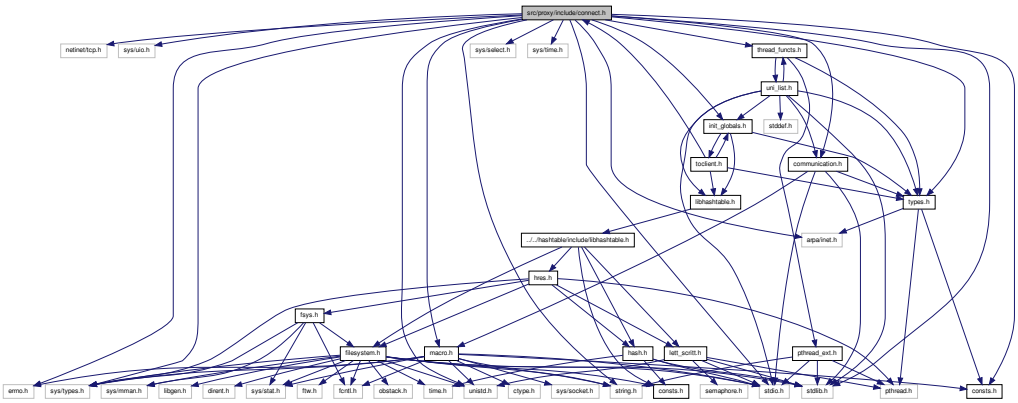
Parametri

buf,:	Buffer contenente la risorsa
len,:	Lunghezza della risorsa del buffer

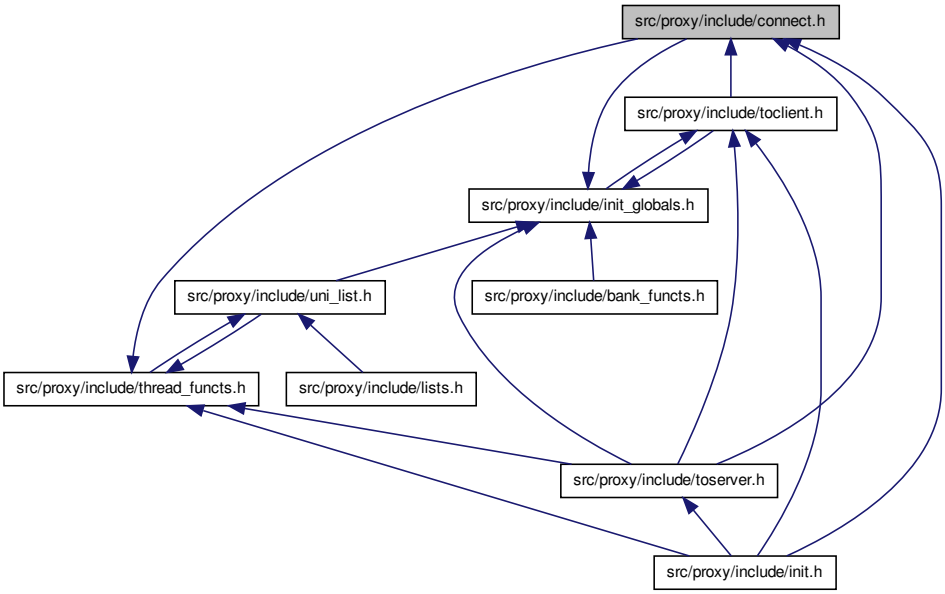
4.20 Riferimenti per il file src/proxy/include/connect.h

```
#include <netinet/tcp.h>
#include <sys/uio.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/select.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include "init_globals.h"
#include "consts.h"
#include "macro.h"
#include "types.h"
#include "communication.h"
#include "thread_functs.h"
```

Grafo delle dipendenze di inclusione per connect.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define MAX_TENTATIVI 10

Funzioni

- void invia_client_termina (struct connection ∗local, char ∗res)
- void fallisci_tutto (struct connection ∗ptr)
- int session (char ∗obtained_remote, char ∗resourcepath, int client)

4.20.1 Documentazione delle definizioni

4.20.1.1 #define MAX_TENTATIVI 10

4.20.2 Documentazione delle funzioni

4.20.2.1 void fallisci_tutto (struct connection ∗ *ptr*)

fallisci_tutto(): Chiude definitivamente la connessione, ed eventualmente effettua l'inoltro della richiesta tramite lettura in cache.

Parametri

ptr,:	Parametro della connessione con il server

4.20.2.2 void invia_client_termina (struct connection ∗ *local,* char ∗ *res*)

invia_client_termina(): Effettua l'invio al client della risorsa ottenuta

Parametri

local,:	Stato di connessione con il server
res,:	Risorsa da inviare al client

4.20.2.3 int session (char ∗ *obtained_remote,* char ∗ *resourcepath,* int *client*)

session(): Effettua la connessione con un server, fornendo la risposta ad un dato client.

Parametri

obtained_remote,:	Buffer di memorizzazione della risposta
resourcepath,:	Percorso della risorsa remota
client,:	FileDescriptor del client

Restituisce

Restituisce 1 in caso di fallimento, altrimenti 0

session(): Instaura una sessione completa con il server, comprensiva della gestione dei tentativi

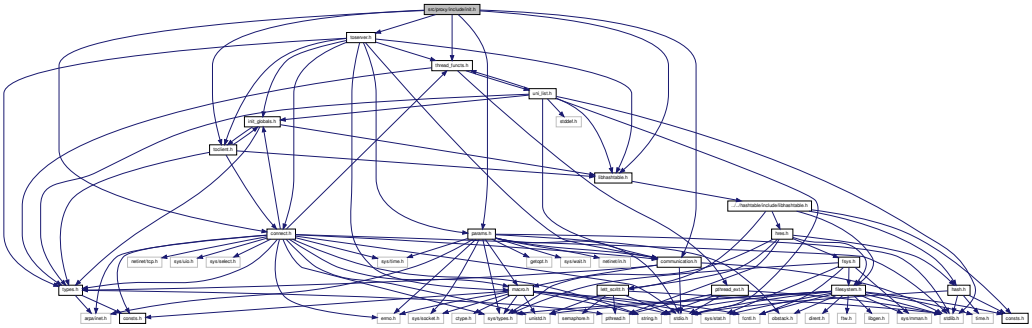
Parametri

obtained_remote,:	Buffer all'interno del quale verrà memorizzata la risorsa
resourcepath,:	Percorso remoto della risorsa da ottenere
client,:	Socket del client

4.21 Riferimenti per il file src/proxy/include/init.h

```
#include "communication.h"
#include "thread_functs.h"
#include "libhashtable.h"
#include "toserver.h"
#include "toclient.h"
#include "connect.h"
#include "params.h"
```

Grafo delle dipendenze di inclusione per init.h:



Funzioni

- int start_threads ()
- int init_proxy_clientside (struct proxy *data)
- request * handle_client_request (struct proxy *data, int *fd_client)

4.21.1 Documentazione delle funzioni

4.21.1.1 request* handle_client_request (struct proxy * *data*, int * *fd_client*)

handle_client_request(): Accetta una richiesta dal client

Parametri

data,:	Informazione di stato del proxy
fd_client,:	Se la procedura termina con successo, setta il fd del client

4.21.1.2 int init_proxy_clientside (struct proxy * *data*)

init_proxy_clientside(): Predispone il proxy ad accettare richieste lato client

init_proxy_clientside(): Effettua l'inizializzazione della connessione, in modo da essere in grado di accettare le richieste

Parametri

data,:	Puntatore allo stato di connessione con il client

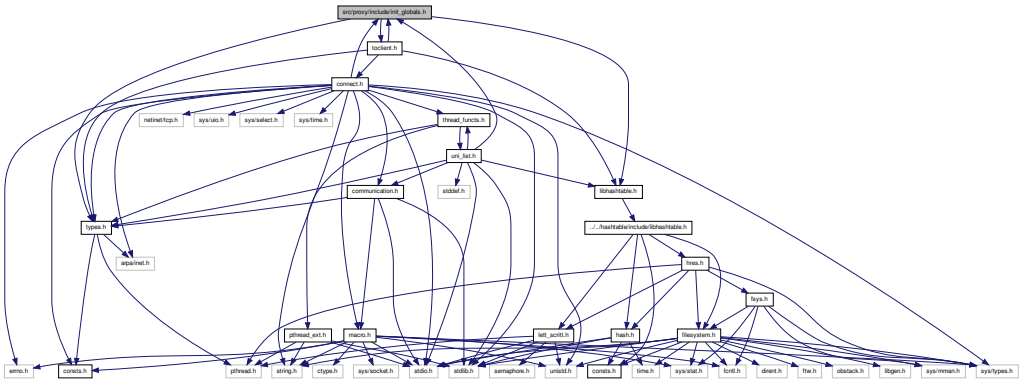
4.21.1.3 int start_threads ( )

start_threads(): Funzione di inizializzazione dei threads e delle Pool

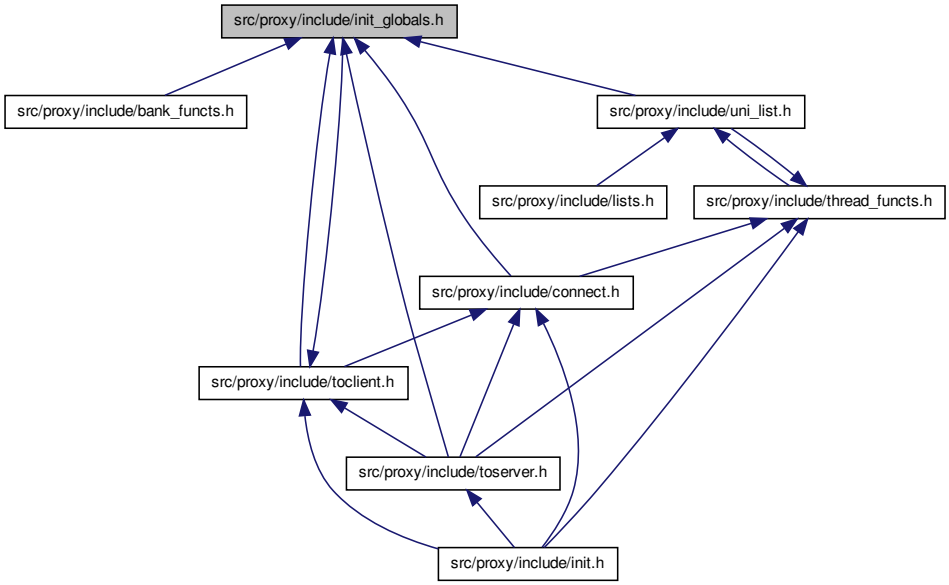
4.22 Riferimenti per il file src/proxy/include/init_globals.h

```
#include "types.h"
#include "toclient.h"
#include "libhashtable.h"
```

Grafo delle dipendenze di inclusione per init_globals.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Variabili

- SCHEDULING serverPool
- SCHEDULING prefetchPool
- SCHEDULING cachePool
- struct proxy global
- ARGS_ONE prefetch_arg
- HRES cache_hash_table

4.22.1 Documentazione delle variabili

4.22.1.1 HRES cache_hash_table

4.22.1.2 SCHEDULING cachePool

4.22.1.3 struct proxy global

4.22.1.4 ARGS_ONE prefetch_arg

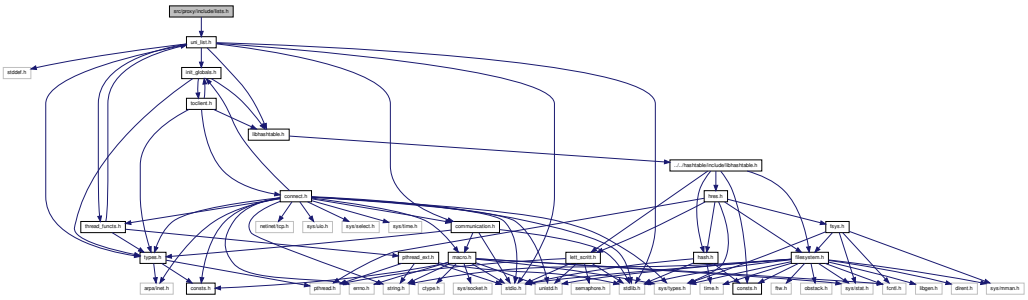
4.22.1.5 SCHEDULING prefetchPool

4.22.1.6 SCHEDULING serverPool

4.23 Riferimenti per il file src/proxy/include/lists.h

```
#include <uni_list.h>
```

Grafo delle dipendenze di inclusione per lists.h:



Strutture dati

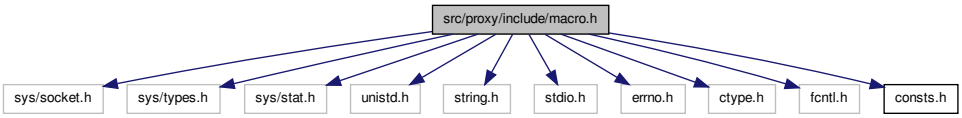
- struct list_ptrint

4.24 Riferimenti per il file src/proxy/include/macro.h

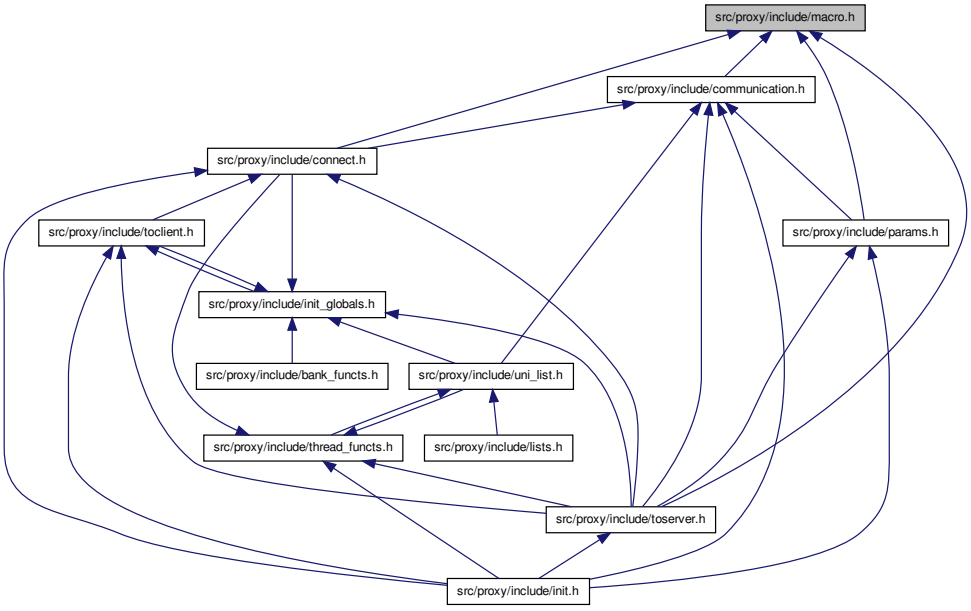
```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
```

```
#include <errno.h>
#include <ctype.h>
#include <fcntl.h>
#include "consts.h"
```

Grafo delle dipendenze di inclusione per macro.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define CPrintf(fg_color,...)
- #define ONCONDRET1(cond) if (cond) return 1
- #define kassert(a,...) while(0)

- #define SET_TIMEOUT(timeout)

4.24.1 Documentazione delle definizioni

4.24.1.1 #define CPrintf(*fg_color, ...*)

Valore:

```
{
        /*      set color mode  */
        printf( "%s%d;%d%s", START_COLOR, BOLD, (fg_color),     END_COLOR);

        /*      print original message  */
        printf( __VA_ARGS__);

        /*      unset color mode        */
        printf( "%s", RESET_COLOR);
}
```

CPrintf() Wrapper for a colored printf()

NOTES: --background won't change

4.24.1.2 #define kassert(*a, ...*) while(0)

KASSERT(): Funzione di asserzione: se la condizione (a) non è verificata, stampa il file e la linea dell'occorrenza dell'errore. I parametri successivi sono come quelli della printf, ed accetta quindi un formato con le variabili.

Parametri

a,:	condizione che non si deve verificare per generare l'errore

4.24.1.3 #define ONCONDRET1(*cond*) if (cond) return 1

4.24.1.4 #define SET_TIMEOUT(*timeout*)

Valore:

```
{
        (timeout)->tv_sec               =       MAX_TIME;
```

```
        (timeout)->tv_usec              =       MAX_uTIME;
}
```

SET_TIMEOUT() Per automatizzare il settaggio del timeout

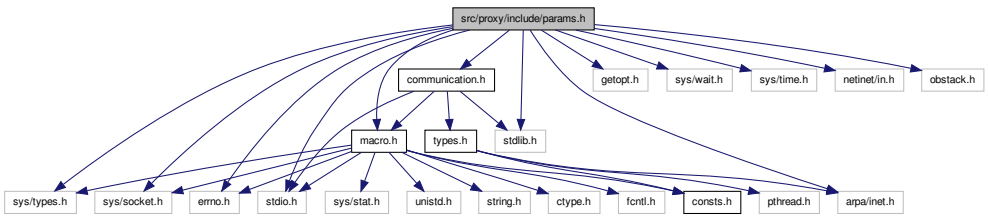
Parametri

timeout,:	indica il tempo da resettare

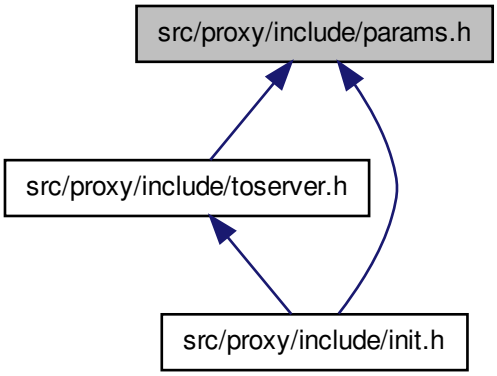
4.25 Riferimenti per il file src/proxy/include/params.h

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <errno.h>
#include <arpa/inet.h>
#include <obstack.h>
#include "macro.h"
#include "communication.h"
```

Grafo delle dipendenze di inclusione per params.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define obstack_chunk_alloc malloc
- #define obstack_chunk_free free
- #define IS_EOOPT(ptr) (((((ptr)->name)||((ptr)->has_arg)||((ptr)->flag)||((ptr)->val))==0) ? 1 : 0)
- #define GETOPT_L(argc, argv, longk, optptr, w) getopt_long(argc,argv,w,longk,optptr)
- #define WHILE_GETOPT_L(argc, argv, longk, optptr, val, w) while(((val)=GETOPT_L(argc,argv,longk,optptr,w))!=-1)
- #define LONGNAME(struc, opt) ((struc)[opt].name)
- #define HAS_ARG (optarg)
- #define HAS_OTHER_ARGS(argc) (optind<(argc))
- #define REMAINING_ARGS(argc) ((argc)-optind+1)
- #define GET_NEXT_OPTIND(argc, argv) (HAS_OTHER_ARGS(argc) ? ((argv)[optind++]) : ((char*)0))
- #define START_CYCLE(argc, argv, longk)
- #define END_CYCLE } } free(allocando); } while(0)

Funzioni

- int init_params (int argc, char *argv[], struct proxy *params)

4.25.1 Documentazione delle definizioni

4.25.1.1 #define END_CYCLE } } free(allocando); } while(0)

END_CYCLE: Loop termination

4.25.1.2 #define GET_NEXT_OPTIND(*argc, argv*) (HAS_OTHER_ARGS(argc) ? ((argv)[optind++]) : ((char*)0))

GET_NEXT_OPTIND(): If there is another arg, returns its char pointer

4.25.1.3 #define GETOPT_L(*argc, argv, longk, optptr, w*) getopt_long(argc,argv,w,longk,optptr)

GETOPT_L(): Calls getopt_long using mkshortopt()

4.25.1.4 #define HAS_ARG (optarg)

HAS_ARG: The pointer to the value of the argument

4.25.1.5 #define HAS_OTHER_ARGS(*argc*) (optind<(argc))

HAS_OTHER_ARGS(): At the end of the cycle, returns if there are some other arguments

4.25.1.6 #define IS_EOOPT(*ptr*) (((((ptr)->name)||((ptr)->has_arg)||((ptr)->flag)||((ptr)->val))==0) ? 1 : 0)

IS_EOOPT(): Macro utilizzate per la definizione della lettura dei parametri

4.25.1.7 #define LONGNAME(*struc, opt*) ((struc)[opt].name)

LONGNAME(): Obtains the long name of the current pace

4.25.1.8 #define obstack_chunk_alloc malloc

4.25.1.9 #define obstack_chunk_free free

4.25.1.10 #define REMAINING_ARGS(*argc*) ((argc)-optind+1)

REMAINING_ARGS(): Counts the remaining argc(s)

4.25.1.11 #define START_CYCLE(*argc, argv, longk*)

Valore:

```
do {\
                                                    int option_index = 0;\
                                                    int c;\
                                                    char* allocando = mkshortopt(long
      k);\
                                                    WHILE_GETOPT_L(argc,argv,longk,&o
      ption_index,c,allocando) { \
                                                    switch(c) {
```

START_CYCLE(): Defines the cycles to retrieve args.

4.25.1.12 #define WHILE_GETOPT_L(*argc, argv, longk, optptr, val, w*) while(((val)=GETOPT_L(argc,argv,longk,optptr,w))!=-1)

WHILE_GETOPT_L(): Does the matching using mkshortopt()

4.25.2 Documentazione delle funzioni

4.25.2.1 int init_params (int *argc*, char * *argv[]*, struct proxy * *params*)

init_params() effettua il parsing dei parametri e setta l'indirizzo ip e porta da far utilizzare al proxy

init_params(): Effettua il parsing dei parametri accettati dal main.

Parametri

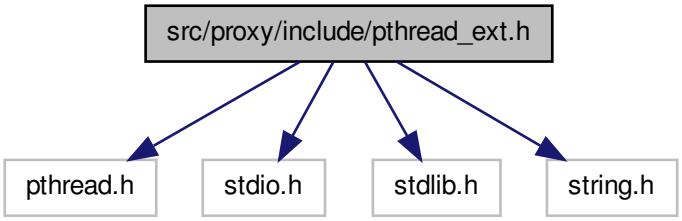
args,:	lunghezza dei parametri
argv,:	Parametri di lunghezza dichiarata nel parametro precedente
params,:	Puntatore allo stato di connessione lato client

INIZIO DEL RICONOSCIMENTO DEI PARAMETRI

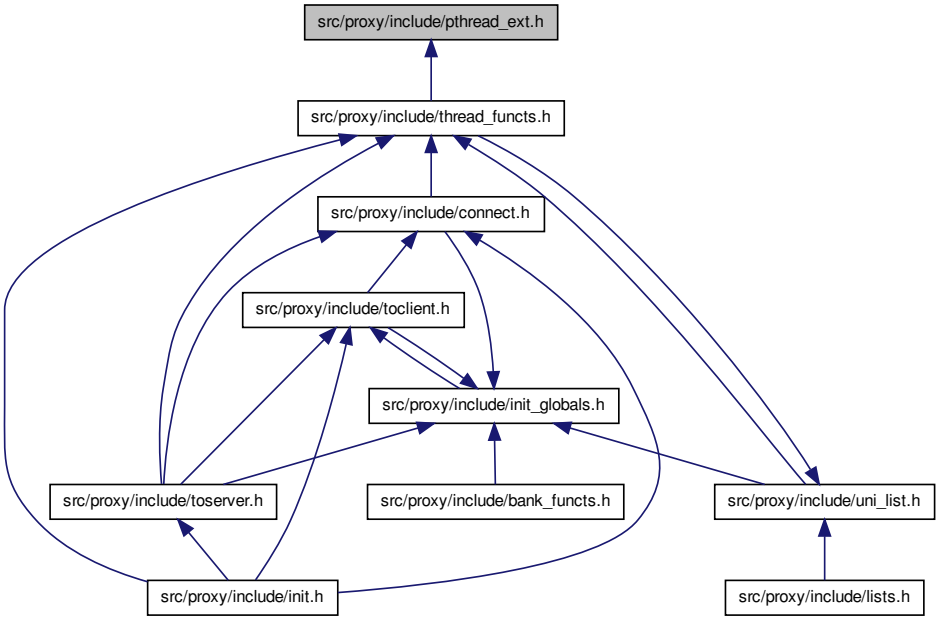
4.26 Riferimenti per il file src/proxy/include/pthread_ext.h

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

Grafo delle dipendenze di inclusione per pthread_ext.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define mutex_init(x)

- #define cond_init(x)

4.26.1 Documentazione delle definizioni

4.26.1.1 #define cond_init(*x*)

Valore:

```
do {                                        \
                        int rtn = pthread_cond_init(x,NULL);    \
                        if (rtn) { \
                                fprintf(stderr,"Error while init condition: %s\n"
      , (char*)strerror(rtn));          \
                                exit(1);                        \
                        }                                       \
                       } while(0)
```

cond_init(): Funzione per la semplificazione della inizializzazione delle condizioni

4.26.1.2 #define mutex_init(*x*)

Valore:

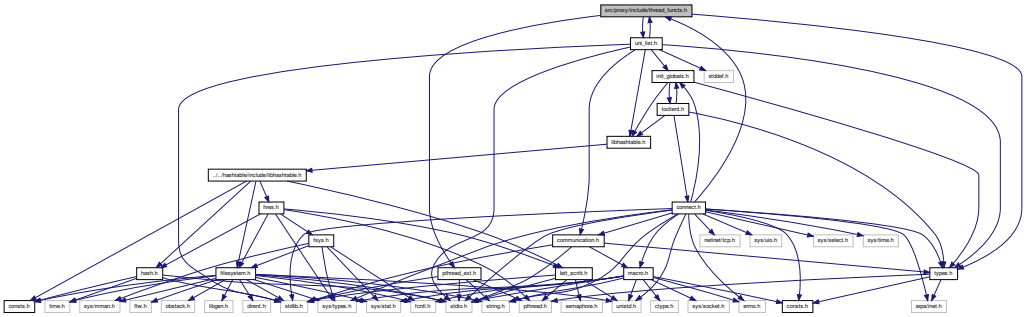
```
do {                                        \
                        int rtn  = pthread_mutex_init(x,NULL); \
                        if (rtn) {\
                                fprintf(stderr,"Error while init mutex: %s\n", (c
      har*)strerror(rtn));     \
                                exit(1);                        \
                        }                                       \
                       } while(0)
```

mutex_init(): Macro per la semplificazione della inizializzazione dei mutex

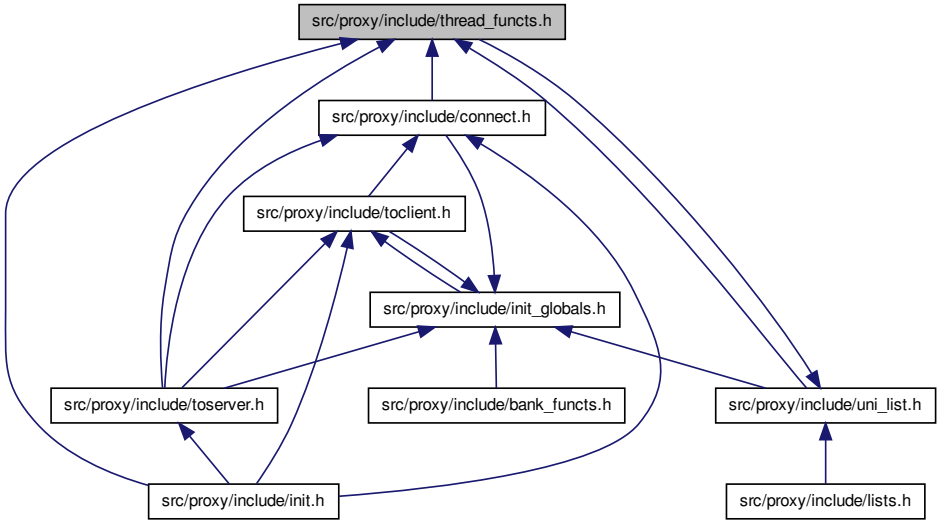
4.27 Riferimenti per il file src/proxy/include/thread_functs.h

```
#include "pthread_ext.h"
#include "uni_list.h"
#include "types.h"
```

Grafo delle dipendenze di inclusione per thread_functs.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define FINEDIMONDO 42

Funzioni

- int init_scheduler (SCHEDULING ∗timbracartellino)

- void ∗ thread_memento (void ∗todo)
- void main_add_job (SCHEDULING ∗todo, void(∗func)(), void ∗args, char priorize)

4.27.1 Documentazione delle definizioni

4.27.1.1 #define FINEDIMONDO 42

4.27.2 Documentazione delle funzioni

4.27.2.1 int init_scheduler (SCHEDULING ∗ *timbracartellino*)

init_scheduler(): Inizializza la struttura dati dello scheduler, predisponendo i mutex ed inizializzando i valori

init_scheduler(): Inizializza la struttura dati dello scheduler, predisponendo i mutex ed inizializzando i valori:

Parametri

timbracartellino,:	Puntatore alla Pool associata.

4.27.2.2 void main_add_job (SCHEDULING ∗ *todo,* void(∗)() *func,* void ∗ *args,* char *priorize*)

main_add_job(): Questa funzione effettua l'inserimento di dati nella coda dei JOB

Parametri

todo,:	Puntatore allo scheduler
func,:	Puntatore alla funzione da eseguire
args,:	Puntatore agli argomenti della lista
priorize,:	Se settato a vero, inserisce in testa, altrimenti in coda

4.27.2.3 void∗ thread_memento (void ∗ *x*)

thread_memento(): Questa funzione effettua il controllo se esiste un lavoro da eseguire o meno

thread_memento(): Questa funzione effettua il controllo se esiste un lavoro da eseguire o meno

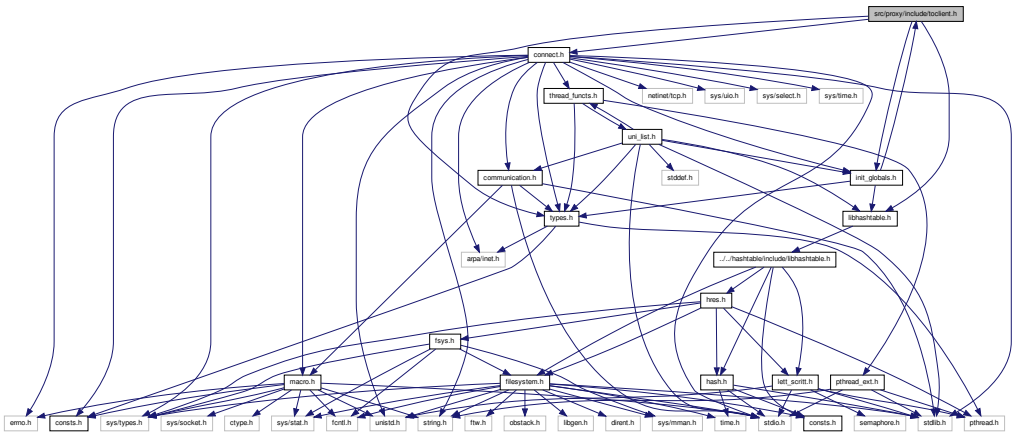
Parametri

x,:	parametro di tipo ARGS_ONE

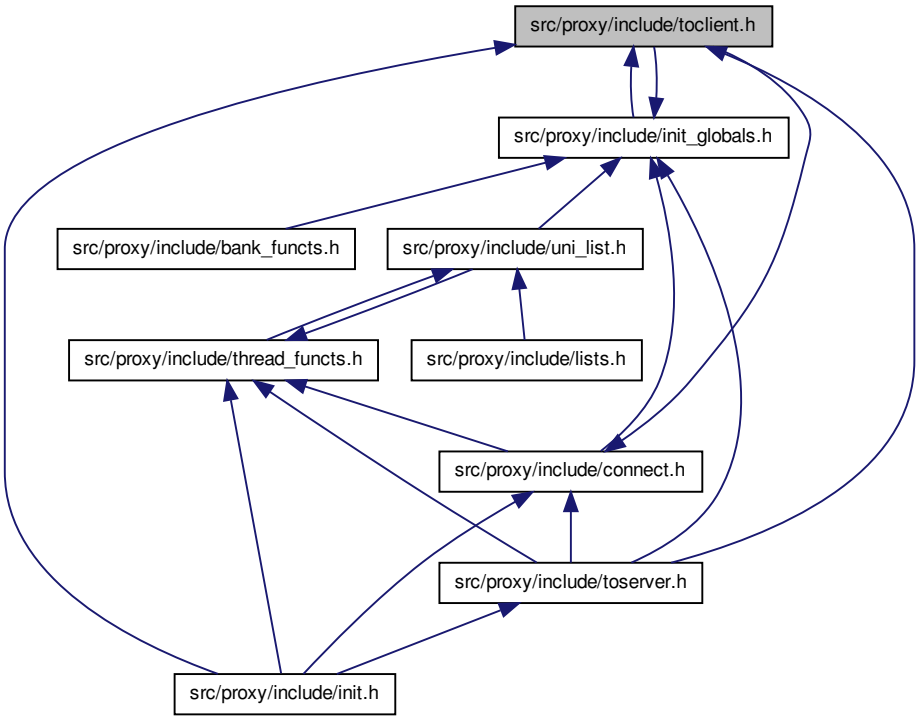
4.28 Riferimenti per il file src/proxy/include/toclient.h

```
#include "libhashtable.h"
#include "init_globals.h"
#include "types.h"
#include "connect.h"
```

Grafo delle dipendenze di inclusione per toclient.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Strutture dati

- struct request_cache

Funzioni

- void toclient (int whoami, void *arg)

4.28.1 Documentazione delle funzioni

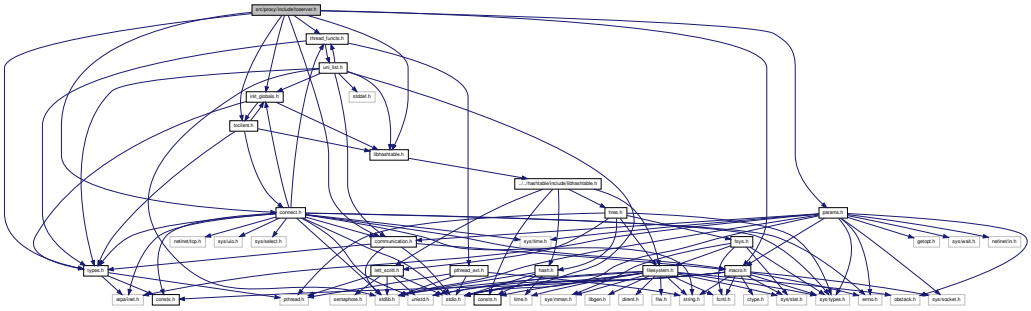
4.28.1.1 void toclient (int *whoami,* void * *arg*)

toclient(): Inoltra la risposta al client, tramite lettura della cache

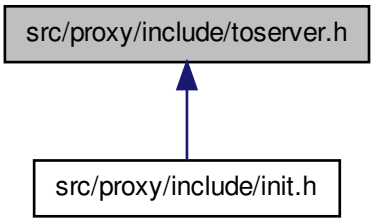
4.29 Riferimenti per il file src/proxy/include/toserver.h

```
#include "communication.h"
#include "thread_functs.h"
#include "libhashtable.h"
#include "init_globals.h"
#include "toclient.h"
#include "connect.h"
#include "params.h"
#include "types.h"
#include "macro.h"
```

Grafo delle dipendenze di inclusione per toserver.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Funzioni

- int fetch (char *buffer, char *remote_path, int fdclient)
- void handle_serverPool (int whoami, msg *params)
- void handle_prefetchPool (int whoami, struct reslist *prefetch_res)

4.29.1 Documentazione delle funzioni

4.29.1.1 int fetch (char * *buffer*, char * *remote_path*, int *fdclient*)

fetch(): Questa funzione prende una risorsa remota e, in caso di successo, effettua il salvataggio in cache

Parametri

buffer,:	Area di memoria dove verrà salvata la risorsa
remote_-path,:	Percorso remoto associato alla risorsa
fdclient,:	FileDescriptor lato client: se -1, non invia nulla

Restituisce

Restituisce 1 in caso di errore, altrimenti 0

fetch(): Richiesta della risorsa ed eventuale memorizzazione nel filesystem

4.29.1.2 void handle_prefetchPool (int *whoami*, struct reslist * *prefetch_res*)

handle_prefetchPool(): Gestione del prefetching

Parametri

whoami,:	Indicatore del thread corrente
prefetch_-res:Indicazior	della risorsa da ottenere

4.29.1.3 void handle_serverPool (int *whoami*, msg * *params*)

handle_serverPool(): Gestione delle richieste verso il server, direttamente dal client

handle_serverPool(): Gestione della richiesta client

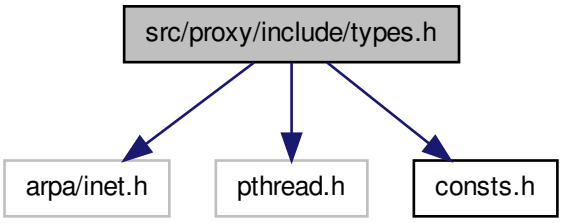
Parametri

whoami,:	Indicatore del thread corrente
params,:	Parametri indicanti la richiesta

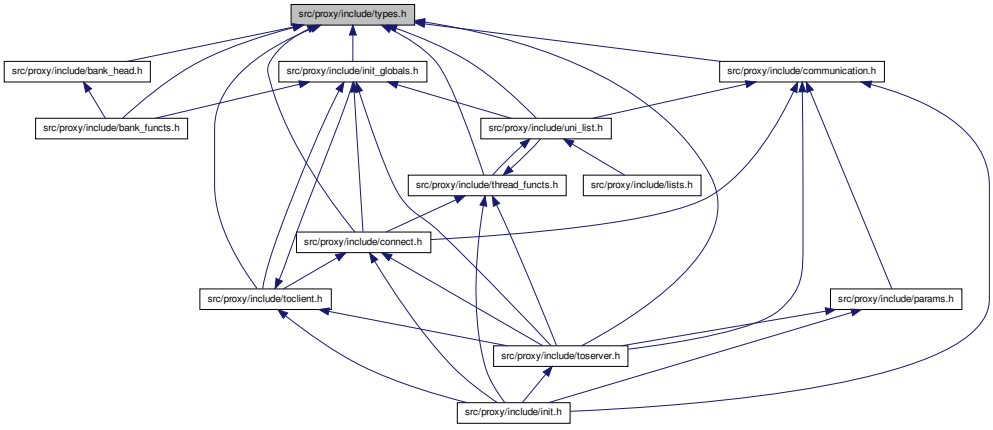
4.30 Riferimenti per il file src/proxy/include/types.h

```
#include <arpa/inet.h>
#include <pthread.h>
#include "consts.h"
```

Grafo delle dipendenze di inclusione per types.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Strutture dati

- struct request
- struct msg

- struct reslist
- struct __mialista__
- struct JOB
- struct JOB_QUEUE
- struct SCHEDULING
- struct ARGS_ONE
- struct connection
- struct proxy

Ridefinizioni di tipo (typedef)

- typedef struct __mialista__ LIST

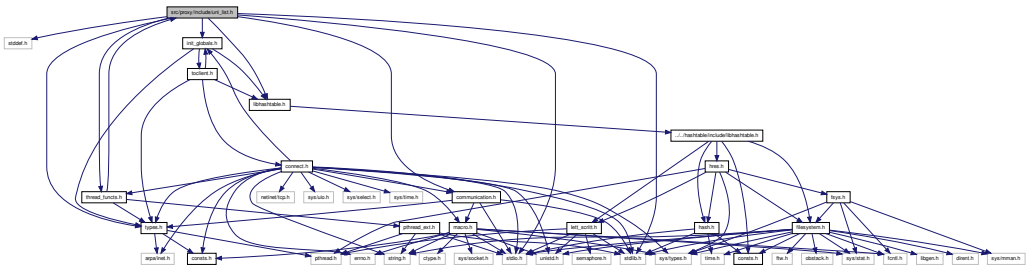
4.30.1 Documentazione delle ridefinizioni di tipo (typedef)

4.30.1.1 typedef struct __mialista__ LIST

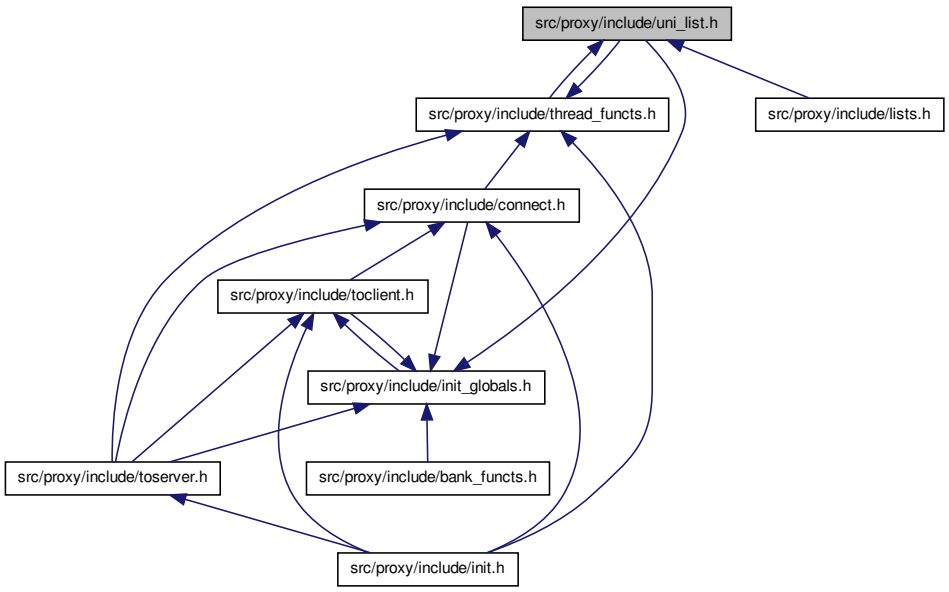
4.31 Riferimenti per il file src/proxy/include/uni_list.h

```
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include "libhashtable.h"
#include "thread_functs.h"
#include "types.h"
#include "init_globals.h"
#include "communication.h"
```

Grafo delle dipendenze di inclusione per uni_list.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Definizioni

- #define init_list_head(head) ((LIST∗)head)->next = NULL
- #define container_of2(ptr, type, member, ret)

Funzioni

- LIST ∗ new_list (void)
- LIST ∗ dequeue (LIST ∗∗ptr)
- void enqueue (LIST ∗ptr, LIST ∗include)
- LIST ∗ remove_elem (LIST ∗ptr, LIST ∗elem)
- void init_job_queue (LIST ∗tmp)
- JOB_QUEUE ∗ dequeue_job (LIST ∗ptr)
- void run_job (int k, JOB_QUEUE ∗tmp)
- JOB_QUEUE ∗ remove_job (LIST ∗ptr, JOB_QUEUE ∗elem)
- JOB_QUEUE ∗ alloc_new_job (void(∗procedura)(), void ∗args)
- void enqueue_job (LIST ∗ptr, JOB_QUEUE ∗include)
- void push_job (LIST ∗ptr, JOB_QUEUE ∗include)
- char exists_job_res (LIST ∗ptr, char ∗path)
- void update_list (char ∗buffer, void(∗func)(), int curr_pos, int MAX_POS, int bufferlow, int bufferhight)

4.31.1 Documentazione delle definizioni

4.31.1.1 #define container_of2(*ptr, type, member, ret*)

Valore:

```
/*do {          */                                          \
        /*const typeof( ((type *)0)->member ) *__mptr = (ptr);  \
        ret = (type *)( (char *)__mptr - offsetof(type,member) );*/\
        ret = (type *)( (char *)ptr - offsetof(type,member) )
```

container_of2(): Ottiene il contenitore della struttura dati puntatore

4.31.1.2 #define init_list_head(*head*) ((LIST*)head)->next = NULL

init_list_head(): Inizializzazione della struttura dati di tipo lista

4.31.2 Documentazione delle funzioni

4.31.2.1 JOB_QUEUE* alloc_new_job (void(*)() *procedura,* void * *args*)

alloc_new_job(): Dato un puntatore generico a funzione e un puntatore ad una struttura dati dell'argomento che la funzione accetta, viene generato un nuovo elemento della lista

4.31.2.2 LIST* dequeue (LIST ** *ptr*)

dequeue(): Disaccoda il primo elemento dalla lista

4.31.2.3 JOB_QUEUE* dequeue_job (LIST * *ptr*)

dequeue_job_queue(): Disaccoda il primo elemento dalla lista ed esegue il primo lavoro

4.31.2.4 void enqueue (LIST * *ptr,* LIST * *include*)

enqueue(): Accoda un elemento particolare della lista

4.31.2.5 void enqueue_job (LIST * *ptr,* JOB_QUEUE * *include*)

enqueue_job(): Accoda un elemento particolare della lista

4.31.2.6 char exists_job_res (LIST * *ptr,* char * *path*)

exists_job_res(): Effettua la verifica che sia già stata inoltrata tale richiesta di download remoto

4.31.2.7 void init_job_queue (LIST * *tmp*)

init_job_queue(): Creazione di un nuovo elemento lista

LISTE PER LA CODA DEI JOB init_job_queue(): Creazione di un nuovo elemento lista

4.31.2.8 LIST* new_list (void)

4.31.2.9 void push_job (LIST * *ptr,* JOB_QUEUE * *include*)

push_job(): Mette in testa della lista l'elemento

4.31.2.10 LIST* remove_elem (LIST * *ptr,* LIST * *elem*)

remove_elem(): rimuove un dato elemento dalla lista

4.31.2.11 JOB_QUEUE* remove_job (LIST * *ptr,* JOB_QUEUE * *elem*)

remove_job(): rimuove un dato elemento dalla lista

4.31.2.12 void run_job (int *k,* JOB_QUEUE * *tmp*)

4.31.2.13 void update_list (char * *buffer,* void(*)() *func,* int *curr_pos,* int *MAX_POS,* int *bufferlow,* int *bufferhight*)

update_list(): Dato un buffer dove è contenuta la parte di risorsa che è stata scaricata, aggiunge alla lista res_list le risorse solamente se queste non sono già presenti in cache o se non stanno per essere scaricate.

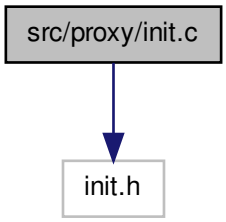
Parametri

buffer,:	buffer dove è contenuta tutta la risorsa remota
func,:	funzione associata all'esecuzione della risorsa
curr_pos,:	identifica il livello corrente nell'albero di download di prefetching
MAX_POS,:	identifica la posizione massima invalicabile, oltre la quale non si effettuerà più download
bufferlow,:	Dimensione bassa del buffer da leggere
bufferhight,:	Dimensione alta del buffer da leggere

4.32 Riferimenti per il file src/proxy/init.c

```
#include "init.h"
```

Grafo delle dipendenze di inclusione per init.c:



Funzioni

- int start_threads ()
- int init_proxy_clientside (struct proxy *data)
- request * handle_client_request (struct proxy *data, int *fd_client)
- int main (int argc, char **argv)

Variabili

- pthread_attr_t pt_attr
- pthread_t server [4]
- pthread_t prefetch
- pthread_t cache
- SCHEDULING serverPool
- SCHEDULING prefetchPool
- SCHEDULING cachePool
- struct proxy global
- ARGS_ONE prefetch_arg
- ARGS_ONE cache_arg
- ARGS_ONE server_thread_args [4]
- HRES cache_hash_table

4.32.1 Documentazione delle funzioni

4.32.1.1 request* handle_client_request (struct proxy * *data*, int * *fd_client*)

handle_client_request(): Accetta una richiesta dal client

Parametri

data,:	Informazione di stato del proxy
fd_client,:	Se la procedura termina con successo, setta il fd del client

4.32.1.2 int init_proxy_clientside (struct proxy * *data*)

init_proxy_clientside(): Effettua l'inizializzazione della connessione, in modo da essere in grado di accettare le richieste

Parametri

data,:	Puntatore allo stato di connessione con il client

4.32.1.3 int main (int *argc,* char ** *argv*)

4.32.1.4 int start_threads ( )

start_threads(): Funzione di inizializzazione dei threads e delle Pool

4.32.2 Documentazione delle variabili

4.32.2.1 pthread_t cache

4.32.2.2 ARGS_ONE cache_arg

4.32.2.3 HRES cache_hash_table

4.32.2.4 SCHEDULING cachePool

4.32.2.5 struct proxy global

4.32.2.6 pthread_t prefetch

4.32.2.7 ARGS_ONE prefetch_arg

4.32.2.8 SCHEDULING prefetchPool

4.32.2.9 pthread_attr_t pt_attr

4.32.2.10 pthread_t server[4]

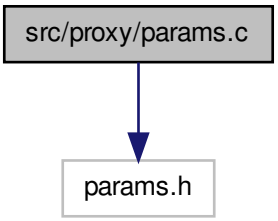
4.32.2.11 ARGS_ONE server_thread_args[4]

4.32.2.12 SCHEDULING serverPool

4.33 Riferimenti per il file src/proxy/params.c

```
#include "params.h"
```

Grafo delle dipendenze di inclusione per params.c:



Funzioni

- char * mkshortopt (struct option *longopt)
- void help ()
- int init_params (int argc, char *argv[ ], struct proxy *params)

4.33.1 Documentazione delle funzioni

4.33.1.1 void help ( )

help(): Stampa in output l'help sull'utilizzo del programma

4.33.1.2 int init_params (int *argc,* char * *argv[],* struct proxy * *params*)

init_params(): Effettua il parsing dei parametri accettati dal main.

Parametri

args,:	lunghezza dei parametri
argv,:	Parametri di lunghezza dichiarata nel parametro precedente
params,:	Puntatore allo stato di connessione lato client

INIZIO DEL RICONOSCIMENTO DEI PARAMETRI

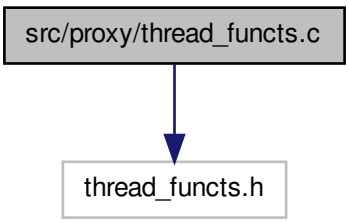
4.33.1.3 char * mkshortopt (struct option * *longopt*)

mkshortopt(): Given the struct option data structure, it returns the correspondent shortopts char, by allocation of a string

4.34 Riferimenti per il file src/proxy/thread_functs.c

```
#include "thread_functs.h"
```

Grafo delle dipendenze di inclusione per thread_functs.c:



Funzioni

- int init_scheduler (SCHEDULING ∗timbracartellino)
- void ∗ thread_memento (void ∗x)
- void main_add_job (SCHEDULING ∗todo, void(∗func)(), void ∗args, char priorize)

Variabili

- int oldoutptfd

4.34.1 Documentazione delle funzioni

4.34.1.1 int init_scheduler (SCHEDULING ∗ *timbracartellino*)

init_scheduler(): Inizializza la struttura dati dello scheduler, predisponendo i mutex ed inizializzando i valori:

Parametri

timbracartellino,:	Puntatore alla Pool associata.

4.34.1.2 void main_add_job (SCHEDULING ∗ *todo,* void(∗)() *func,* void ∗ *args,* char *priorize*)

main_add_job(): Questa funzione effettua l'inserimento di dati nella coda dei JOB

Parametri

todo,:	Puntatore allo scheduler
func,:	Puntatore alla funzione da eseguire
args,:	Puntatore agli argomenti della lista
priorize,:	Se settato a vero, inserisce in testa, altrimenti in coda

4.34.1.3 void* thread_memento (void * *x*)

thread_memento(): Questa funzione effettua il controllo se esiste un lavoro da eseguire o meno

Parametri

x,:	parametro di tipo ARGS_ONE

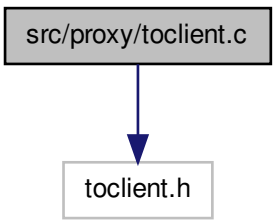
4.34.2 Documentazione delle variabili

4.34.2.1 int oldoutptfd

4.35 Riferimenti per il file src/proxy/toclient.c

```
#include "toclient.h"
```

Grafo delle dipendenze di inclusione per toclient.c:



Funzioni

- void toclient (int whoami, void *arg)

4.35.1 Documentazione delle funzioni

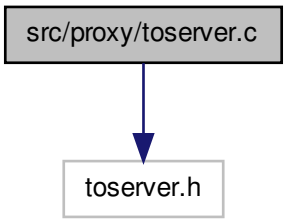
4.35.1.1 void toclient (int *whoami,* void * *arg*)

toclient(): Inoltra la risposta al client, tramite lettura della cache

4.36 Riferimenti per il file src/proxy/toserver.c

```
#include "toserver.h"
```

Grafo delle dipendenze di inclusione per toserver.c:



Funzioni

- int fetch (char *buffer, char *remote_path, int fdclient)
- void handle_serverPool (int whoami, msg *params)
- void handle_prefetchPool (int whoami, struct reslist *prefetch_res)

4.36.1 Documentazione delle funzioni

4.36.1.1 int fetch (char * *buffer,* char * *remote_path,* int *fdclient*)

fetch(): Richiesta della risorsa ed eventuale memorizzazione nel filesystem

4.36.1.2 void handle_prefetchPool (int *whoami,* struct reslist * *prefetch_res*)

handle_prefetchPool(): Gestione del prefetching

Parametri

whoami,:	Indicatore del thread corrente

prefetch_-res:Indicazior	della risorsa da ottenere

4.36.1.3 void handle_serverPool (int *whoami,* msg * *params*)

handle_serverPool(): Gestione della richiesta client

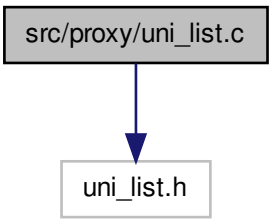
Parametri

whoami,:	Indicatore del thread corrente
params,:	Parametri indicanti la richiesta

4.37 Riferimenti per il file src/proxy/uni_list.c

```
#include "uni_list.h"
```

Grafo delle dipendenze di inclusione per uni_list.c:



Funzioni

- LIST * new (void)
- LIST * dequeue (LIST **ptr)
- LIST * remove_elem (LIST *ptr, LIST *elem)
- void enqueue (LIST *ptr, LIST *include)
- void push (LIST *ptr, LIST *include)
- void init_job_queue (LIST *tmp)
- JOB_QUEUE * dequeue_job (LIST *ptr)
- void run_job (int k, JOB_QUEUE *tmp)
- JOB_QUEUE * remove_job (LIST *ptr, JOB_QUEUE *elem)
- JOB_QUEUE * alloc_new_job (void(*procedura)(), void *args)

- void enqueue_job (LIST ∗ptr, JOB_QUEUE ∗include)
- void push_job (LIST ∗ptr, JOB_QUEUE ∗include)
- char exists_job_res (LIST ∗ptr, char ∗path)
- void update_list (char ∗buffer, void(∗func)(), int curr_pos, int MAX_POS, int bufferlow, int bufferhight)

Variabili

- int oldoutptfd

4.37.1 Documentazione delle funzioni

4.37.1.1 JOB_QUEUE∗ alloc_new_job (void(∗)() *procedura,* void ∗ *args*)

alloc_new_job(): Dato un puntatore generico a funzione e un puntatore ad una struttura dati dell'argomento che la funzione accetta, viene generato un nuovo elemento della lista

4.37.1.2 LIST∗ dequeue (LIST ∗∗ *ptr*)

dequeue(): Disaccoda il primo elemento dalla lista

4.37.1.3 JOB_QUEUE∗ dequeue_job (LIST ∗ *ptr*)

dequeue_job_queue(): Disaccoda il primo elemento dalla lista ed esegue il primo lavoro

4.37.1.4 void enqueue (LIST ∗ *ptr,* LIST ∗ *include*)

enqueue(): Accoda un elemento particolare della lista

4.37.1.5 void enqueue_job (LIST ∗ *ptr,* JOB_QUEUE ∗ *include*)

enqueue_job(): Accoda un elemento particolare della lista

4.37.1.6 char exists_job_res (LIST ∗ *ptr,* char ∗ *path*)

exists_job_res(): Effettua la verifica che sia già stata inoltrata tale richiesta di download remoto

4.37.1.7 void init_job_queue (LIST ∗ *tmp*)

LISTE PER LA CODA DEI JOB init_job_queue(): Creazione di un nuovo elemento lista

4.37.1.8 LIST* new (void)

new(): Creazione di un nuovo elemento lista

4.37.1.9 void push (LIST * *ptr,* LIST * *include*)

push(): Effettua l'inserimento con priorità degli elementi

4.37.1.10 void push_job (LIST * *ptr,* JOB_QUEUE * *include*)

push_job(): Mette in testa della lista l'elemento

4.37.1.11 LIST* remove_elem (LIST * *ptr,* LIST * *elem*)

remove_elem(): rimuove un dato elemento dalla lista

4.37.1.12 JOB_QUEUE* remove_job (LIST * *ptr,* JOB_QUEUE * *elem*)

remove_job(): rimuove un dato elemento dalla lista

4.37.1.13 void run_job (int *k,* JOB_QUEUE * *tmp*)

4.37.1.14 void update_list (char * *buffer,* void(*)() *func,* int *curr_pos,* int *MAX_POS,* int *bufferlow,* int *bufferhight*)

update_list(): Dato un buffer dove è contenuta la parte di risorsa che è stata scaricata, aggiunge alla lista res_list le risorse solamente se queste non sono già presenti in cache o se non stanno per essere scaricate.

Parametri

buffer,:	buffer dove è contenuta tutta la risorsa remota
func,:	funzione associata all'esecuzione della risorsa
curr_pos,:	identifica il livello corrente nell'albero di download di prefetching
MAX_POS,:	identifica la posizione massima invalicabile, oltre la quale non si effettuerà più download
bufferlow,:	Dimensione bassa del buffer da leggere
bufferhight,:	Dimensione alta del buffer da leggere

4.37.2 Documentazione delle variabili

4.37.2.1 int oldoutptfd

Indice analitico