

Proxy(c)2012

Generato da Doxygen 1.7.3

Fri Feb 17 2012 14:42:44

Indice

1	Indice delle strutture dati	1
1.1	Strutture dati	1
2	Indice dei file	3
2.1	Elenco dei file	3
3	Documentazione delle classi	5
3.1	Riferimenti per la struct __mialista__	5
3.1.1	Documentazione dei campi	5
3.1.1.1	next	5
3.2	Riferimenti per la struct ARGS_ONE	6
3.2.1	Descrizione dettagliata	7
3.2.2	Documentazione dei campi	7
3.2.2.1	sched	7
3.2.2.2	whoami	7
3.3	Riferimenti per la struct connection	7
3.3.1	Descrizione dettagliata	7
3.3.2	Documentazione dei campi	8
3.3.2.1	expiry	8
3.3.2.2	fd_client	8
3.3.2.3	fd_server	8
3.3.2.4	PATH	8
3.3.2.5	sIP	8
3.3.2.6	sPort	8
3.3.2.7	tentativi	8
3.4	Riferimenti per la struct CS_struct	8
3.4.1	Documentazione dei campi	9
3.4.1.1	attr	9
3.5	Riferimenti per la struct hash_tbl	9
3.5.1	Descrizione dettagliata	9
3.5.2	Documentazione dei campi	9
3.5.2.1	elements	9
3.5.2.2	value	9
3.6	Riferimenti per la struct HRES	9
3.6.1	Documentazione dei campi	10
3.6.1.1	hashtable	10
3.6.1.2	lettscrit	10
3.7	Riferimenti per la struct JOB	11
3.7.1	Descrizione dettagliata	11

3.7.2	Documentazione dei campi	11
3.7.2.1	args	11
3.7.2.2	procedura	11
3.8	Riferimenti per la struct JOB_QUEUE	11
3.8.1	Descrizione dettagliata	12
3.8.2	Documentazione dei campi	12
3.8.2.1	head	12
3.8.2.2	todo	13
3.9	Riferimenti per la struct list_pprint	13
3.9.1	Documentazione dei campi	14
3.9.1.1	elem	14
3.9.1.2	valore	14
3.10	Riferimenti per la struct msg	14
3.10.1	Descrizione dettagliata	15
3.10.2	Documentazione dei campi	15
3.10.2.1	rq	15
3.10.2.2	sock	15
3.11	Riferimenti per la struct proxy	15
3.11.1	Descrizione dettagliata	15
3.11.2	Documentazione dei campi	16
3.11.2.1	fd	16
3.11.2.2	ip	16
3.11.2.3	level	16
3.11.2.4	meStesso	16
3.11.2.5	port	16
3.12	Riferimenti per la struct request	16
3.12.1	Descrizione dettagliata	16
3.12.2	Documentazione dei campi	16
3.12.2.1	PATH	16
3.12.2.2	rangehigh	17
3.12.2.3	rangelow	17
3.12.2.4	TYPE	17
3.13	Riferimenti per la struct request_cache	17
3.13.1	Documentazione dei campi	17
3.13.1.1	elemen	17
3.13.1.2	fd_client	17
3.13.1.3	fd_file	17
3.13.1.4	remote	17
3.14	Riferimenti per la struct reslist	17
3.14.1	Descrizione dettagliata	18
3.14.2	Documentazione dei campi	18
3.14.2.1	prefetch_num	18
3.14.2.2	remote_path	18
3.15	Riferimenti per la struct SCHEDULING	18
3.15.1	Descrizione dettagliata	19
3.15.2	Documentazione dei campi	20
3.15.2.1	non_empty	20
3.15.2.2	planning	20
3.15.2.3	qlock	20
3.15.2.4	qsize	20

4	Documentazione dei file	21
4.1	Riferimenti per il file <code>src/hashtable/filesystem.c</code>	21
4.1.1	Documentazione delle funzioni	22
4.1.1.1	<code>file_exists</code>	22
4.1.1.2	<code>folder_empty</code>	22
4.1.1.3	<code>obtain_local</code>	22
4.1.1.4	<code>re_new_resource</code>	22
4.1.1.5	<code>recursiveDelete</code>	22
4.1.1.6	<code>resource_exists</code>	23
4.1.1.7	<code>resource_remove</code>	23
4.2	Riferimenti per il file <code>src/hashtable/fsys.c</code>	23
4.2.1	Documentazione delle funzioni	24
4.2.1.1	<code>FileClose</code>	24
4.2.1.2	<code>FileCreate</code>	24
4.2.1.3	<code>FileOpen</code>	25
4.3	Riferimenti per il file <code>src/hashtable/hash.c</code>	25
4.3.1	Documentazione delle definizioni	26
4.3.1.1	<code>SEED</code>	26
4.3.2	Documentazione delle funzioni	26
4.3.2.1	<code>hash</code>	26
4.3.2.2	<code>hash_exists</code>	26
4.3.2.3	<code>hash_insert</code>	26
4.3.2.4	<code>hash_occupied</code>	26
4.3.2.5	<code>hash_remove</code>	26
4.3.2.6	<code>hashb</code>	26
4.3.2.7	<code>init_hashtbl</code>	26
4.4	Riferimenti per il file <code>src/hashtable/hres.c</code>	27
4.4.1	Documentazione delle funzioni	27
4.4.1.1	<code>close_cached_file</code>	27
4.4.1.2	<code>handle_file_filesystem</code>	27
4.4.1.3	<code>init_hres</code>	28
4.4.1.4	<code>is_inhash</code>	28
4.4.1.5	<code>map_cached_file</code>	28
4.4.1.6	<code>parseHead_time</code>	28
4.5	Riferimenti per il file <code>src/hashtable/include/consts.h</code>	29
4.5.1	Documentazione delle definizioni	30
4.5.1.1	<code>HASH_SIZE</code>	30
4.5.1.2	<code>MAX_FILE</code>	30
4.5.1.3	<code>MAX_HRES</code>	30
4.5.1.4	<code>MOD</code>	30
4.5.1.5	<code>MULT1</code>	30
4.5.1.6	<code>MULT2</code>	30
4.5.1.7	<code>NHASH1</code>	30
4.5.1.8	<code>NHASH2</code>	30
4.6	Riferimenti per il file <code>src/proxy/include/consts.h</code>	30
4.6.1	Documentazione delle definizioni	33
4.6.1.1	<code>BLACK</code>	33
4.6.1.2	<code>BLUE</code>	33
4.6.1.3	<code>BOLD</code>	33
4.6.1.4	<code>CYAN</code>	33

4.6.1.5	END_COLOR	33
4.6.1.6	GET	33
4.6.1.7	GREEN	33
4.6.1.8	INF	33
4.6.1.9	INFO	33
4.6.1.10	INTERVAL_NOT_FOUND	33
4.6.1.11	LIMIT_FILE	33
4.6.1.12	MAGENTA	33
4.6.1.13	MAX_BUFF	33
4.6.1.14	MAX_FILE	33
4.6.1.15	MAX_PATH	33
4.6.1.16	MAX_TIME	33
4.6.1.17	MAX_TIMESELECT	33
4.6.1.18	MAX_uTIME	33
4.6.1.19	MAXCONN	33
4.6.1.20	MAXIDREF	33
4.6.1.21	MAXREF	33
4.6.1.22	MAXREQUEST	33
4.6.1.23	NO_MORE_THREAD	33
4.6.1.24	NOT_FOUND	33
4.6.1.25	OK	33
4.6.1.26	OK_INFO	33
4.6.1.27	OK_RANGE	33
4.6.1.28	PROXY_IP	33
4.6.1.29	PROXY_PORT	33
4.6.1.30	RED	33
4.6.1.31	RESET_COLOR	33
4.6.1.32	SOCKET_ERROR	33
4.6.1.33	START_COLOR	33
4.6.1.34	UNKNOWN_ERROR	33
4.6.1.35	UNKNOWN_ERROR_SIZE	33
4.6.1.36	WHITE	33
4.6.1.37	WRONG_REQUEST	33
4.6.1.38	YELLOW	33
4.7	Riferimenti per il file src/hashtable/include/filesystem.h	33
4.7.1	Documentazione delle definizioni	36
4.7.1.1	_GNU_SOURCE	36
4.7.1.2	cache_free	36
4.7.1.3	close_map	36
4.7.1.4	EXPIRED	36
4.7.1.5	FILE_SIZE	36
4.7.1.6	NOW	36
4.7.1.7	PAST	36
4.7.2	Documentazione delle funzioni	36
4.7.2.1	file_exists	36
4.7.2.2	folder_empty	37
4.7.2.3	obtain_local	37
4.7.2.4	re_new_resource	37
4.7.2.5	recursiveDelete	37
4.7.2.6	resource_exists	38

4.7.2.7	resource_remove	38
4.8	Riferimenti per il file src/hashtable/include/fsys.h	38
4.8.1	Documentazione delle funzioni	40
4.8.1.1	FileClose	40
4.8.1.2	FileCreate	40
4.8.1.3	FileOpen	40
4.9	Riferimenti per il file src/hashtable/include/hash.h	41
4.9.1	Documentazione delle definizioni	43
4.9.1.1	_BSD_SOURCE	43
4.9.2	Documentazione delle funzioni	43
4.9.2.1	hash	43
4.9.2.2	hash_exists	43
4.9.2.3	hash_insert	43
4.9.2.4	hash_occupied	43
4.9.2.5	hash_remove	43
4.9.2.6	hashb	43
4.9.2.7	init_hashtbl	43
4.10	Riferimenti per il file src/hashtable/include/hres.h	43
4.10.1	Documentazione delle funzioni	45
4.10.1.1	close_cached_file	45
4.10.1.2	handle_file_filesystem	45
4.10.1.3	init_hres	45
4.10.1.4	is_inhash	45
4.10.1.5	map_cached_file	46
4.11	Riferimenti per il file src/hashtable/include/lett_scritt.h	46
4.11.1	Documentazione delle definizioni	48
4.11.1.1	P	48
4.11.1.2	V	48
4.11.1.3	WREAD	48
4.11.1.4	WWRITE	48
4.11.2	Documentazione delle ridefinizioni di tipo (typedef)	48
4.11.2.1	ANYTYPE	48
4.11.3	Documentazione delle funzioni	48
4.11.3.1	init_struct	48
4.11.3.2	struct_do_read	48
4.11.3.3	struct_do_write	48
4.11.3.4	struct_end_read	49
4.11.3.5	struct_end_write	49
4.12	Riferimenti per il file src/hashtable/include/libhashtable.h	49
4.13	Riferimenti per il file src/proxy/include/libhashtable.h	50
4.14	Riferimenti per il file src/hashtable/lett_scritt.c	51
4.14.1	Documentazione delle funzioni	52
4.14.1.1	init_struct	52
4.14.1.2	struct_do_read	52
4.14.1.3	struct_do_write	52
4.14.1.4	struct_end_read	52
4.14.1.5	struct_end_write	52
4.15	Riferimenti per il file src/proxy/communication.c	52
4.15.1	Documentazione delle funzioni	53
4.15.1.1	atreturn	53

4.15.1.2	atreturns	54
4.15.1.3	doreply	54
4.15.1.4	dorequest	54
4.15.1.5	get_resource_alloc	54
4.15.1.6	getfirstlevel_next	55
4.15.1.7	getIP	55
4.15.1.8	getPort	55
4.15.1.9	haspos	55
4.15.1.10	parseHead	55
4.15.1.11	parseRequest	56
4.15.1.12	surfto	56
4.15.2	Documentazione delle variabili	56
4.15.2.1	oldoutptfd	56
4.16	Riferimenti per il file src/proxy/connect.c	56
4.16.1	Documentazione delle definizioni	57
4.16.1.1	INIT_CONNECT	57
4.16.2	Documentazione delle funzioni	57
4.16.2.1	base_read	57
4.16.2.2	establishConnection	57
4.16.2.3	fallisci_tutto	57
4.16.2.4	init_connect_struct	58
4.16.2.5	invia_client_termina	58
4.16.2.6	send_request	58
4.16.2.7	server_base_send	58
4.16.2.8	server_complete_read	59
4.16.2.9	server_complete_send	59
4.16.2.10	session	59
4.17	Riferimenti per il file src/proxy/include/bank_funcs.h	59
4.18	Riferimenti per il file src/proxy/include/bank_head.h	60
4.18.1	Documentazione delle funzioni	61
4.18.1.1	cache_done	61
4.18.1.2	cache_exists	61
4.18.1.3	cache_init	62
4.18.1.4	cache_init_socket	62
4.19	Riferimenti per il file src/proxy/include/communication.h	62
4.19.1	Documentazione delle definizioni	64
4.19.1.1	_GNU_SOURCE	64
4.19.1.2	distance	64
4.19.2	Documentazione delle funzioni	64
4.19.2.1	doreply	64
4.19.2.2	dorequest	64
4.19.2.3	get_resource_alloc	65
4.19.2.4	getIP	65
4.19.2.5	getPort	65
4.19.2.6	parseHead	65
4.19.2.7	parseRequest	66
4.20	Riferimenti per il file src/proxy/include/connect.h	66
4.20.1	Documentazione delle definizioni	68
4.20.1.1	MAX_TENTATIVI	68
4.20.2	Documentazione delle funzioni	68

4.20.2.1	fallisci_tutto	68
4.20.2.2	invia_client_termina	68
4.20.2.3	session	68
4.21	Riferimenti per il file src/proxy/include/init.h	69
4.21.1	Documentazione delle funzioni	69
4.21.1.1	handle_client_request	69
4.21.1.2	init_proxy_clientside	70
4.21.1.3	start_threads	70
4.22	Riferimenti per il file src/proxy/include/init_globals.h	70
4.22.1	Documentazione delle variabili	72
4.22.1.1	cache_hash_table	72
4.22.1.2	cachePool	72
4.22.1.3	global	72
4.22.1.4	prefetch_arg	72
4.22.1.5	prefetchPool	72
4.22.1.6	serverPool	72
4.23	Riferimenti per il file src/proxy/include/lists.h	72
4.24	Riferimenti per il file src/proxy/include/macro.h	72
4.24.1	Documentazione delle definizioni	74
4.24.1.1	CPrintf	74
4.24.1.2	kassert	74
4.24.1.3	ONCONDRET1	74
4.24.1.4	SET_TIMEOUT	74
4.25	Riferimenti per il file src/proxy/include/params.h	75
4.25.1	Documentazione delle definizioni	77
4.25.1.1	END_CYCLE	77
4.25.1.2	GET_NEXT_OPTIND	77
4.25.1.3	GETOPT_L	77
4.25.1.4	HAS_ARG	77
4.25.1.5	HAS_OTHER_ARGS	77
4.25.1.6	IS_EOOPT	77
4.25.1.7	LONGNAME	77
4.25.1.8	obstack_chunk_alloc	77
4.25.1.9	obstack_chunk_free	77
4.25.1.10	REMAINING_ARGS	77
4.25.1.11	START_CYCLE	78
4.25.1.12	WHILE_GETOPT_L	78
4.25.2	Documentazione delle funzioni	78
4.25.2.1	init_params	78
4.26	Riferimenti per il file src/proxy/include/pthread_ext.h	78
4.26.1	Documentazione delle definizioni	80
4.26.1.1	cond_init	80
4.26.1.2	mutex_init	80
4.27	Riferimenti per il file src/proxy/include/thread_funcs.h	80
4.27.1	Documentazione delle definizioni	82
4.27.1.1	FINEDIMONDO	82
4.27.2	Documentazione delle funzioni	82
4.27.2.1	init_scheduler	82
4.27.2.2	main_add_job	82
4.27.2.3	thread_memento	82

4.28	Riferimenti per il file <code>src/proxy/include/toclient.h</code>	83
4.28.1	Documentazione delle funzioni	84
4.28.1.1	<code>toclient</code>	84
4.29	Riferimenti per il file <code>src/proxy/include/toserver.h</code>	85
4.29.1	Documentazione delle funzioni	86
4.29.1.1	<code>fetch</code>	86
4.29.1.2	<code>handle_prefetchPool</code>	86
4.29.1.3	<code>handle_serverPool</code>	86
4.30	Riferimenti per il file <code>src/proxy/include/types.h</code>	87
4.30.1	Documentazione delle ridefinizioni di tipo (<code>typedef</code>)	88
4.30.1.1	<code>LIST</code>	88
4.31	Riferimenti per il file <code>src/proxy/include/uni_list.h</code>	88
4.31.1	Documentazione delle definizioni	90
4.31.1.1	<code>container_of2</code>	90
4.31.1.2	<code>init_list_head</code>	90
4.31.2	Documentazione delle funzioni	90
4.31.2.1	<code>alloc_new_job</code>	90
4.31.2.2	<code>dequeue</code>	90
4.31.2.3	<code>dequeue_job</code>	90
4.31.2.4	<code>enqueue</code>	90
4.31.2.5	<code>enqueue_job</code>	90
4.31.2.6	<code>exists_job_res</code>	90
4.31.2.7	<code>init_job_queue</code>	91
4.31.2.8	<code>new_list</code>	91
4.31.2.9	<code>push_job</code>	91
4.31.2.10	<code>remove_elem</code>	91
4.31.2.11	<code>remove_job</code>	91
4.31.2.12	<code>run_job</code>	91
4.31.2.13	<code>update_list</code>	91
4.32	Riferimenti per il file <code>src/proxy/init.c</code>	91
4.32.1	Documentazione delle funzioni	92
4.32.1.1	<code>handle_client_request</code>	92
4.32.1.2	<code>init_proxy_clientside</code>	93
4.32.1.3	<code>main</code>	93
4.32.1.4	<code>start_threads</code>	93
4.32.2	Documentazione delle variabili	94
4.32.2.1	<code>cache</code>	94
4.32.2.2	<code>cache_arg</code>	94
4.32.2.3	<code>cache_hash_table</code>	94
4.32.2.4	<code>cachePool</code>	94
4.32.2.5	<code>global</code>	94
4.32.2.6	<code>prefetch</code>	94
4.32.2.7	<code>prefetch_arg</code>	94
4.32.2.8	<code>prefetchPool</code>	94
4.32.2.9	<code>pt_attr</code>	94
4.32.2.10	<code>server</code>	94
4.32.2.11	<code>server_thread_args</code>	94
4.32.2.12	<code>serverPool</code>	94
4.33	Riferimenti per il file <code>src/proxy/params.c</code>	94
4.33.1	Documentazione delle funzioni	95

4.33.1.1	help	95
4.33.1.2	init_params	95
4.33.1.3	mkshorpt	95
4.34	Riferimenti per il file src/proxy/thread_funcs.c	95
4.34.1	Documentazione delle funzioni	96
4.34.1.1	init_scheduler	96
4.34.1.2	main_add_job	96
4.34.1.3	thread_memento	97
4.34.2	Documentazione delle variabili	97
4.34.2.1	oldoutptfd	97
4.35	Riferimenti per il file src/proxy/toclient.c	97
4.35.1	Documentazione delle funzioni	98
4.35.1.1	toclient	98
4.36	Riferimenti per il file src/proxy/toserver.c	98
4.36.1	Documentazione delle funzioni	98
4.36.1.1	fetch	98
4.36.1.2	handle_prefetchPool	98
4.36.1.3	handle_serverPool	99
4.37	Riferimenti per il file src/proxy/uni_list.c	99
4.37.1	Documentazione delle funzioni	100
4.37.1.1	alloc_new_job	100
4.37.1.2	dequeue	100
4.37.1.3	dequeue_job	100
4.37.1.4	enqueue	100
4.37.1.5	enqueue_job	100
4.37.1.6	exists_job_res	100
4.37.1.7	init_job_queue	100
4.37.1.8	new	101
4.37.1.9	push	101
4.37.1.10	push_job	101
4.37.1.11	remove_elem	101
4.37.1.12	remove_job	101
4.37.1.13	run_job	101
4.37.1.14	update_list	101
4.37.2	Documentazione delle variabili	101
4.37.2.1	oldoutptfd	101

Capitolo 1

Indice delle strutture dati

1.1 Strutture dati

Queste sono le strutture dati con una loro breve descrizione:

__mialista__	5
ARGS_ONE	6
connection	7
CS_struct	8
hash_tbl	9
HRES	9
JOB	11
JOB_QUEUE	11
list_pprint	13
msg	14
proxy	15
request	16
request_cache	17
reslist	17
SCHEDULING	18

Capitolo 2

Indice dei file

2.1 Elenco dei file

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

src/hashtable/filesystem.c	21
src/hashtable/fsys.c	23
src/hashtable/hash.c	25
src/hashtable/hres.c	27
src/hashtable/lett_scritt.c	51
src/hashtable/include/consts.h	29
src/hashtable/include/filesystem.h	33
src/hashtable/include/fsys.h	38
src/hashtable/include/hash.h	41
src/hashtable/include/hres.h	43
src/hashtable/include/lett_scritt.h	46
src/hashtable/include/libhashtable.h	49
src/proxy/communication.c	52
src/proxy/connect.c	56
src/proxy/init.c	91
src/proxy/params.c	94
src/proxy/thread_funcs.c	95
src/proxy/toclient.c	97
src/proxy/toserver.c	98
src/proxy/uni_list.c	99
src/proxy/include/bank_funcs.h	59
src/proxy/include/bank_head.h	60
src/proxy/include/communication.h	62
src/proxy/include/connect.h	66
src/proxy/include/consts.h	30
src/proxy/include/init.h	69
src/proxy/include/init_globals.h	70
src/proxy/include/libhashtable.h	50
src/proxy/include/lists.h	72

src/proxy/include/ macro.h	72
src/proxy/include/ params.h	75
src/proxy/include/ pthread_ext.h	78
src/proxy/include/ thread_funcs.h	80
src/proxy/include/ toclient.h	83
src/proxy/include/ toserver.h	85
src/proxy/include/ types.h	87
src/proxy/include/ uni_list.h	88

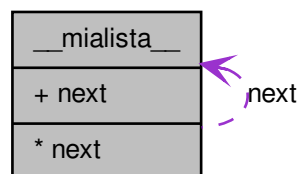
Capitolo 3

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

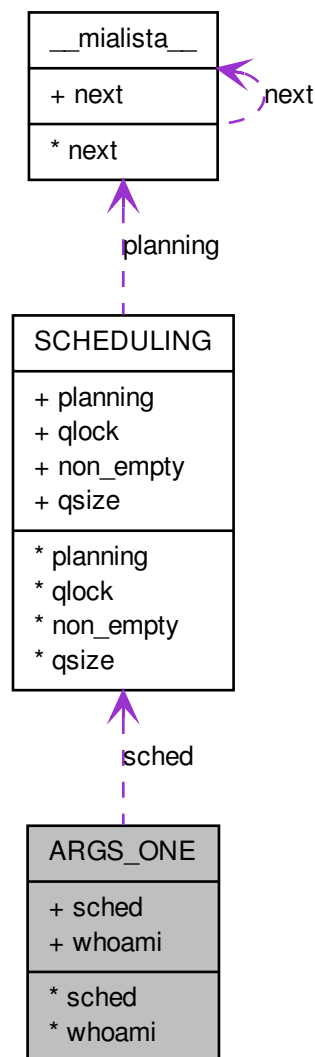
La documentazione per questa struct è stata generata a partire dal seguente file:

- [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

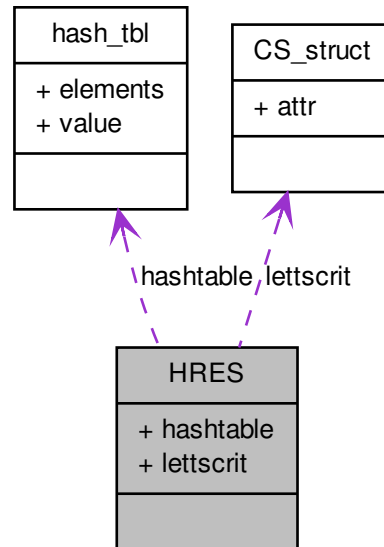
La documentazione per questa struct è stata generata a partire dal seguente file:

- 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

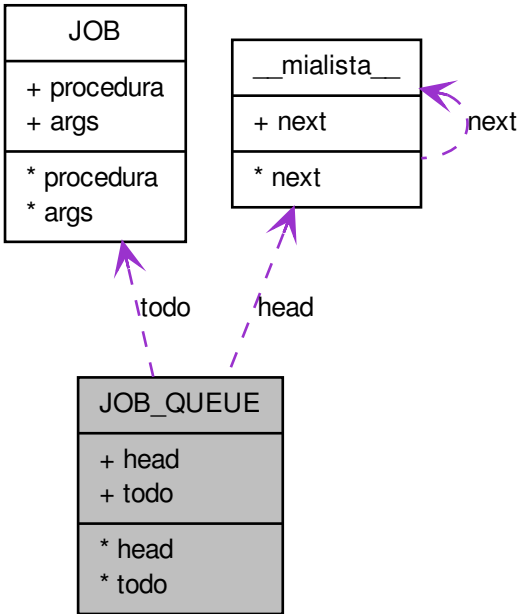
La documentazione per questa struct è stata generata a partire dal seguente file:

- [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

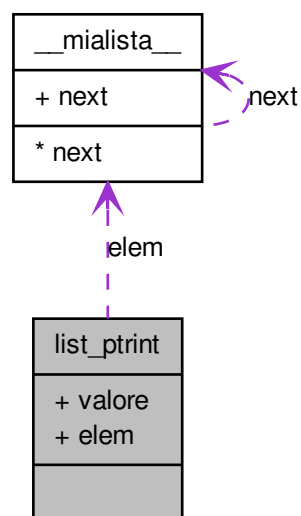
La documentazione per questa struct è stata generata a partire dal seguente file:

- [src/proxy/include/types.h](#)

3.9 Riferimenti per la struct list_pprint

```
#include <lists.h>
```

Diagramma di collaborazione per list_pprint:



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

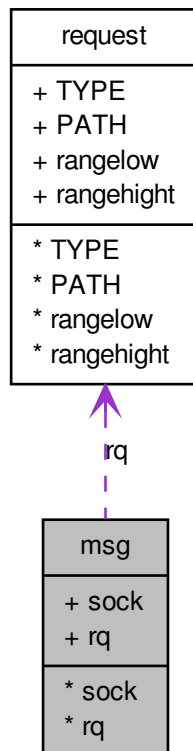
La documentazione per questa struct è stata generata a partire dal seguente file:

- [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 [rangehigh](#)

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::rangehigh

Eventuale higher-range

3.12.2.3 unsigned int request::rangelow

Eventuale lower-range

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

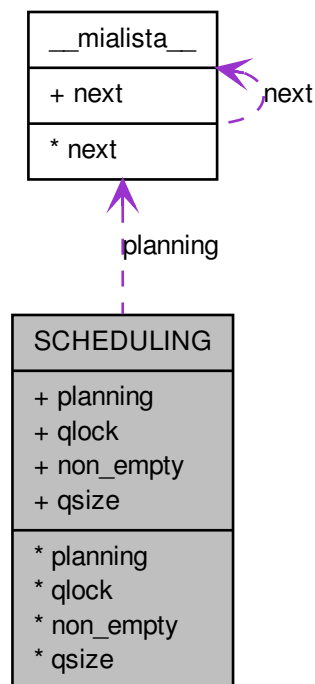
La documentazione per questa struct è stata generata a partire dal seguente file:

- [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](#)

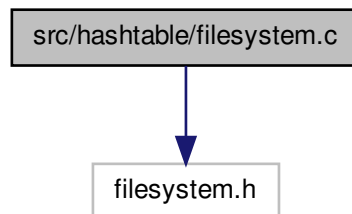
Capitolo 4

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

<i>filename</i> ,:	file locale del quale controllare l'esistenza
<i>time</i> ,:	Conterrà il tempo da Epoch della creazione del file
<i>is_blank</i> ,:	Restituisce, se passato come argomento, se il file è solamente troncato
<i>retbuf</i> ,:	Restituisce lo fstat del file, se esiste.
<i>pfd</i> ,:	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

<i>remote</i> ,:	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

<i>cache_path</i> ,:	Identifica il percorso locale della risorsa
<i>retbuf</i> ,:	Identifica lo stat del file che è stato aperto. Se non NULL viene aggiornato.
<i>pfd</i> ,:	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

<i>dirname,:</i>	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 * pdf)`

[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

<i>remote,:</i>	path remoto
<i>deallocate,:</i>	indica se deallocare l'indirizzo locale della risorsa corrispondente
<i>result,:</i>	verrà scritto in questo puntatore l'esistenza o meno del file
<i>time,:</i>	tempo di creazione della risorsa
<i>blank,:</i>	restituisce se il file è stato troncato o meno (e quindi vuoto)
<i>retbuf,:</i>	restituisce (se non NULL) il puntatore all'fstat del file in cache
<i>pdf,:</i>	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.

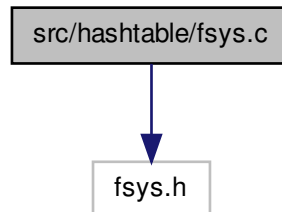
Parametri

<i>path,:</i>	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

<i>update</i> ,:	Puntatore alla risorsa mappata in memoria
<i>fd</i> ,:	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

<i>remote_resource</i> ,:	risorsa remota
<i>expiry</i> ,:	tempo di expire
<i>buffer</i> ,:	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

<i>remote_resource</i> ,:	indicazione della risorsa remota
<i>update</i> ,:	Indicazione dell'area di allocazione del file in cache

Restituisce

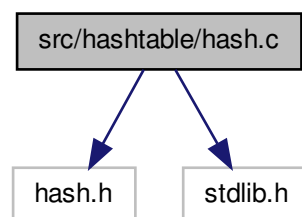
Restituisce 0 in caso di insuccesso, altrimenti il valore del filedescriptor

```
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

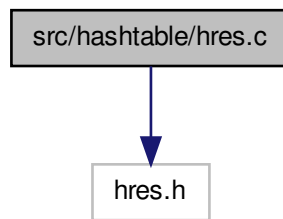
4.3.2.7 void init_hashtbl (struct hash_tbl * p)

`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 filesystem

Parametri

<i>hash</i> ,:	Struttura dati di hashing
<i>remote_file</i> ,:	Indicazione del file remoto di risorsa
<i>res</i> ,:	Indica la risorsa contenente le informazioni, non utilizzata per l'eliminazione del file
<i>creat_oth_delete</i> ,:	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*)

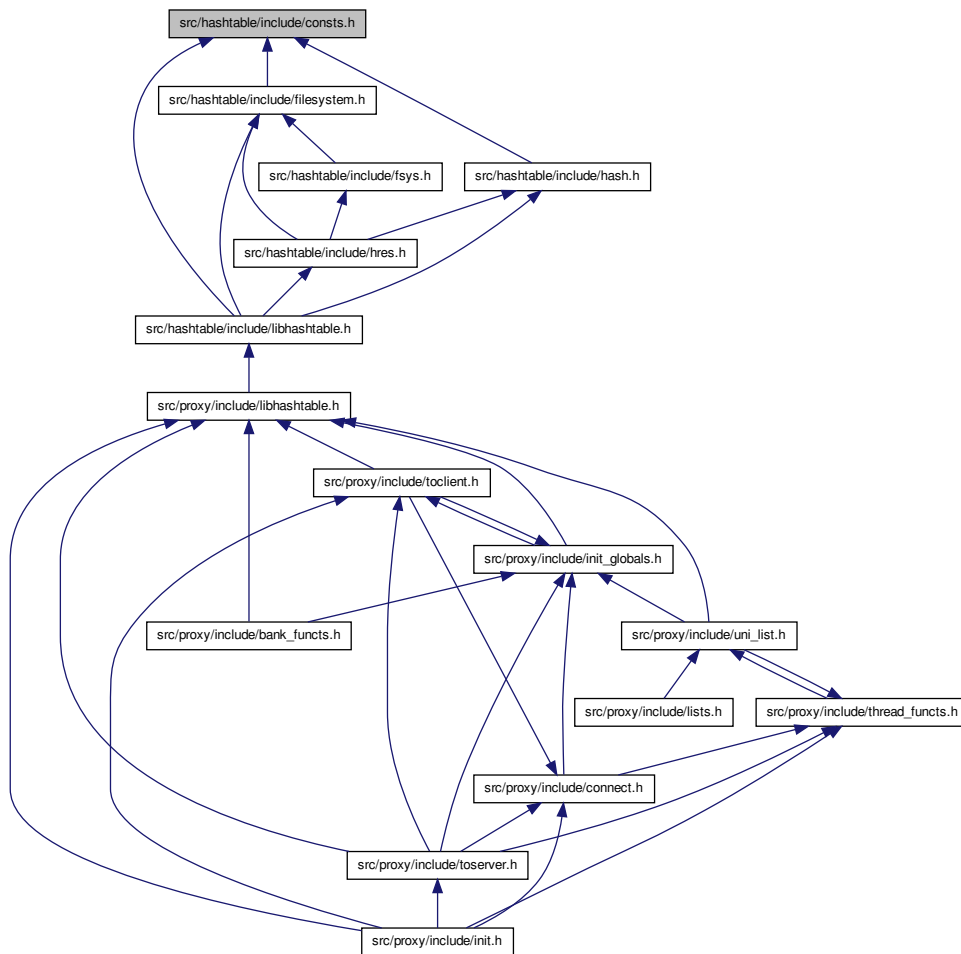
[is_inhash\(\)](#): Controlla se il file è presente in cache, e se esiste effettivamente all'interno del filesystem

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

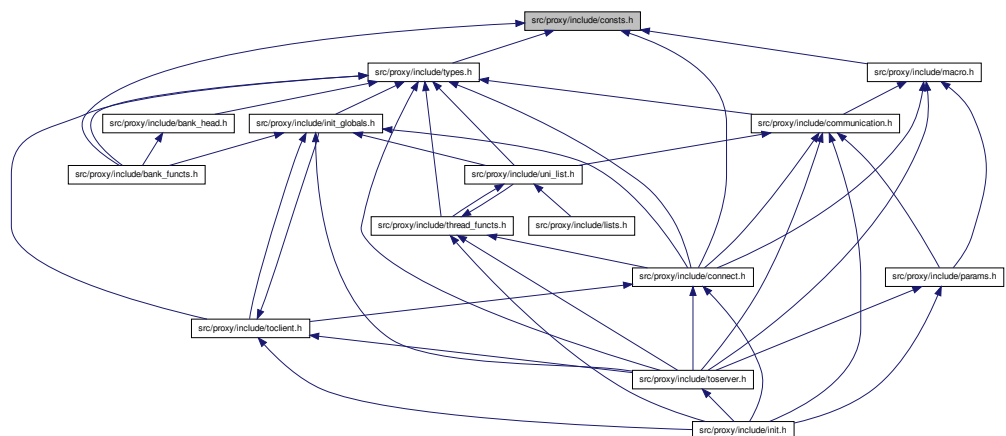
4.5.1.6 #define [MULT2](#) 47

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`

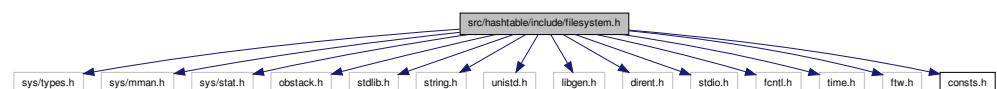
4.6.1.27 `#define OK_RANGE 201`

4.6.1.28 `#define PROXY_IP "127.0.0.1"`

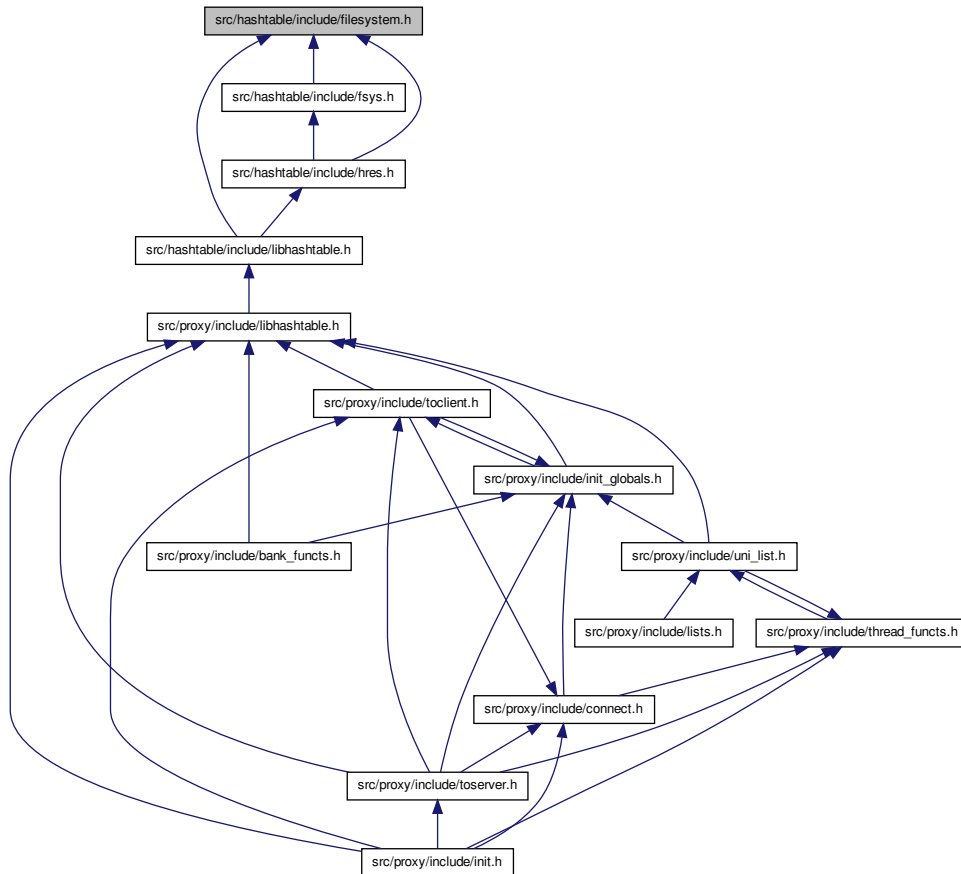
4.6.1.29 `#define PROXY_PORT 55554`

```
#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 *`pdf`)
- char `folder_empty` (char *`dirname`)
- char * `resource_exists` (char *`remote`, int `deallocate`, char *`result`, unsigned int *`time`, char *`blank`, struct stat *`retbuf`, int *`pdf`)
- void `re_new_resource` (char *`cache_path`, struct stat *`retbuf`, int *`pdf`)
- 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

<i>mem,:</i>	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 * pdf)

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

<i>filename,:</i>	file locale del quale controllare l'esistenza
<i>time,:</i>	Conterrà il tempo da Epoch della creazione del file
<i>is_blank,:</i>	Restituisce, se passato come argomento, se il file è solamente troncato
<i>retbuf,:</i>	Restituisce lo fstat del file, se esiste.
<i>pfd,:</i>	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

<i>remote,:</i>	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

<i>cache_path,:</i>	Identifica il percorso locale della risorsa
<i>retbuf,:</i>	Identifica lo stat del file che è stato aperto. Se non NULL viene aggiornato.
<i>pfd,:</i>	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

<i>dirname,:</i>	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

<i>remote,:</i>	path remoto
<i>deallocate,:</i>	indica se deallocare l'indirizzo locale della risorsa corrispondente
<i>result,:</i>	verrà scritto in questo puntatore l'esistenza o meno del file
<i>time,:</i>	tempo di creazione della risorsa
<i>blank,:</i>	restituisce se il file è stato troncato o meno (e quindi vuoto)
<i>retbuf,:</i>	restituisce (se non NULL) il puntatore all'fstat del file in cache
<i>pfd,:</i>	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

<i>remote,:</i>	path remoto
<i>deallocate,:</i>	indica se deallocare l'indirizzo locale della risorsa corrispondente
<i>result,:</i>	verrà scritto in questo puntatore l'esistenza o meno del file
<i>time,:</i>	tempo di creazione della risorsa
<i>blank,:</i>	restituisce se il file è stato troncato o meno (e quindi vuoto)
<i>retbuf,:</i>	restituisce (se non NULL) il puntatore all'fstat del file in cache
<i>pfd,:</i>	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

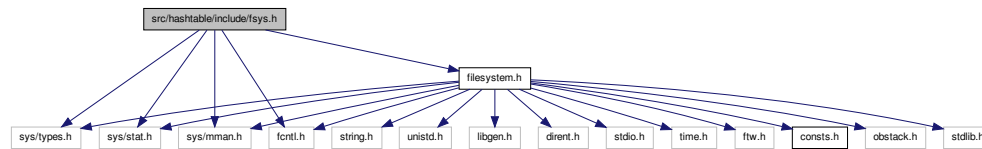
<i>path,:</i>	Risorsa remota
---------------	----------------

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

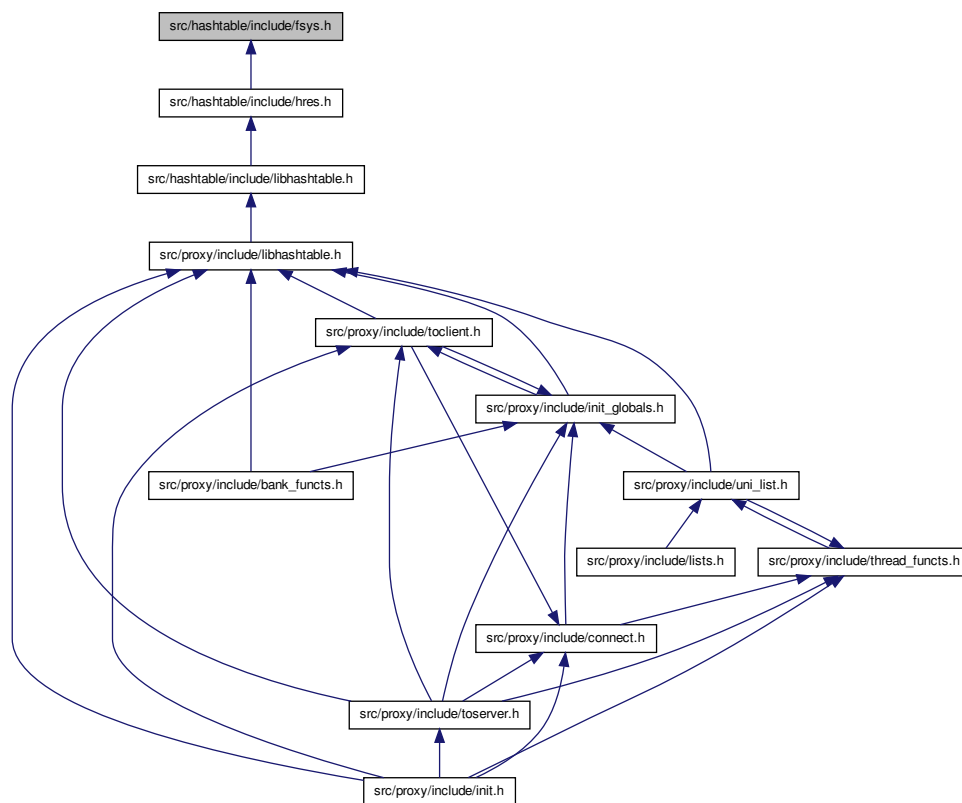
```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
```

```
#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

<i>update</i> ,:	Puntatore alla risorsa mappata in memoria
<i>fd</i> ,:	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

<i>remote_resource</i> ,:	risorsa remota
<i>expiry</i> ,:	tempo di expire
<i>buffer</i> ,:	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

<i>remote_resource</i> ,:	indicazione della risorsa remota
<i>update</i> ,:	Indicazione dell'area di allocazione del file in cache

Restituisce

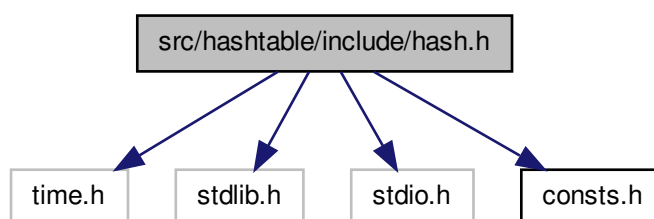
Restituisce 0 in caso di insuccesso, altrimenti il valore del filedescriptor

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

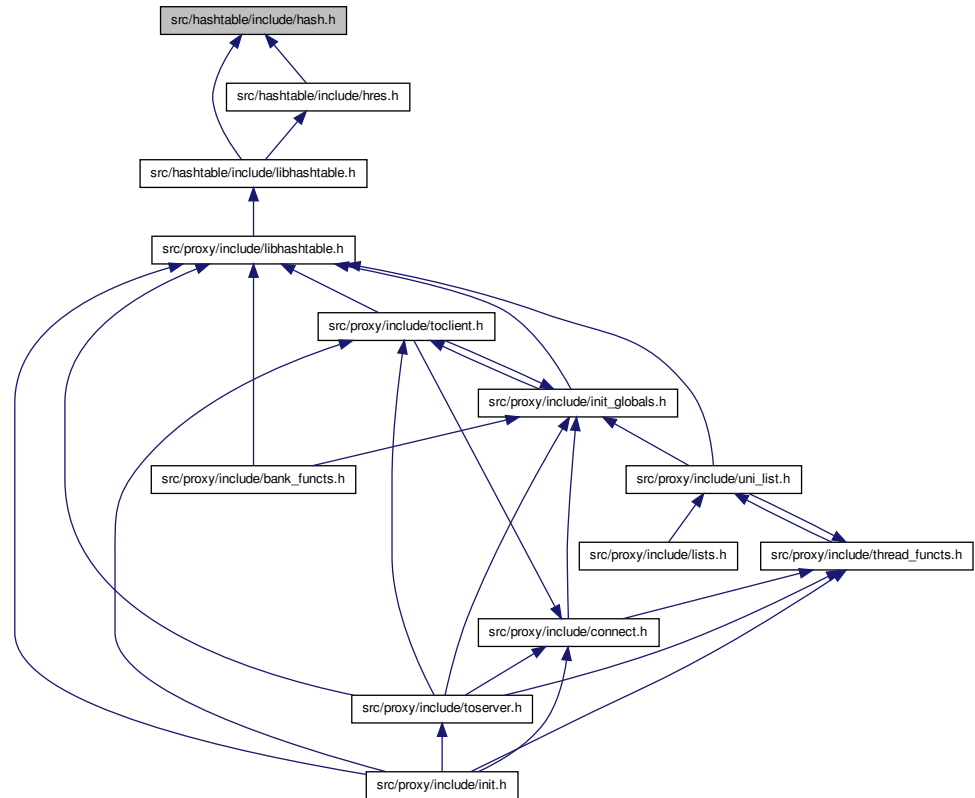
4.9 Riferimenti per il file src/hashtable/include/hash.h

```
#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

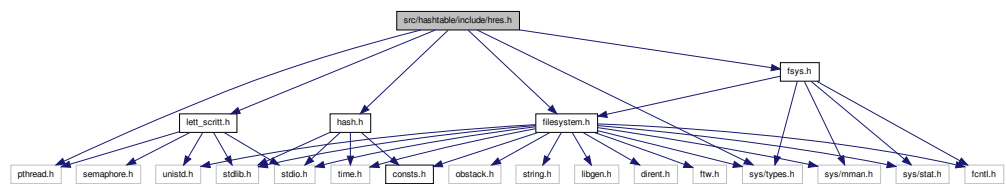
```
#include <sys/types.h>
#include <pthread.h>
#include "lett_scritt.h"
```

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

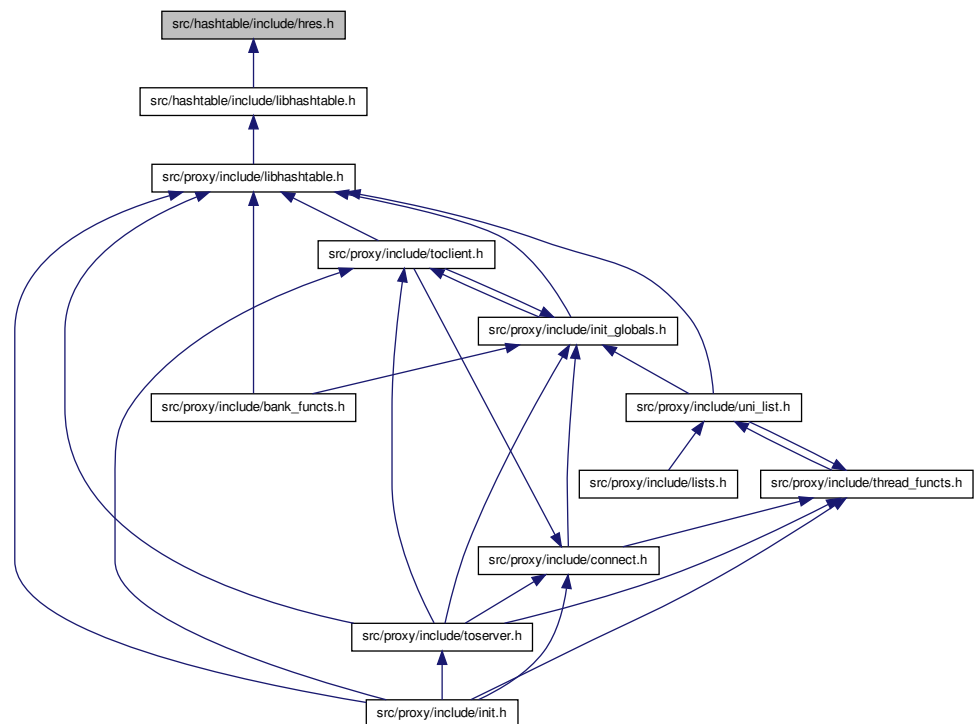
```
#include "fsys.h"
```

```
#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 filesystem

Parametri

<i>hash</i> ,:	Struttura dati di hashing
<i>remote_file</i> ,:	Indicazione del file remoto di risorsa
<i>res</i> ,:	Indica la risorsa contenente le informazioni, non utilizzata per l'eliminazione del file
<i>creat_oth_delete</i> ,:	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 nellahtable (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*)

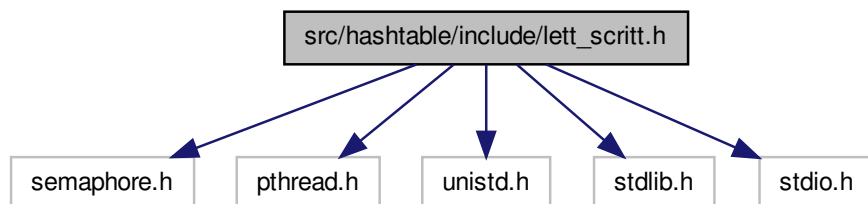
[is_inhash\(\)](#): Controlla se il file è presente in cache, e se esiste effettivamente all'interno del filesystem

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

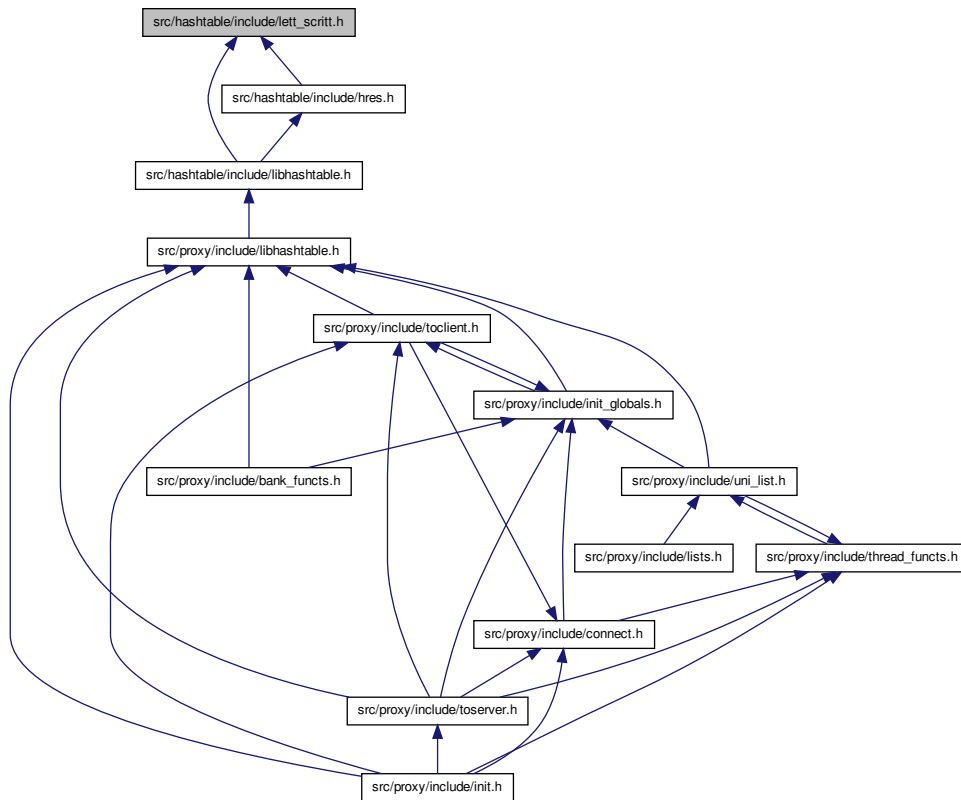
4.11 Riferimenti per il file `src/hashtable/include/lett_scritt.h`

```
#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

<i>data,:</i>	Indica la struttura dati sulla quale si effettua il lock
<i>external,:</i>	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

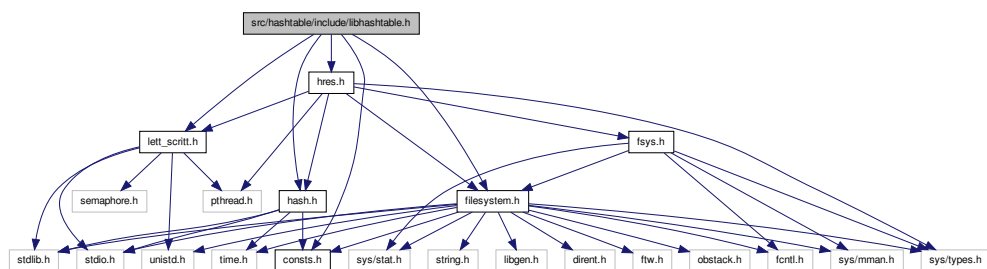
4.11.3.5 void struct_end_write (struct CS_struct * data)

[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"
```

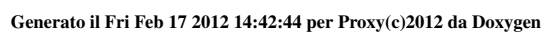
Grafo delle dipendenze di inclusione per libhashtable.h:



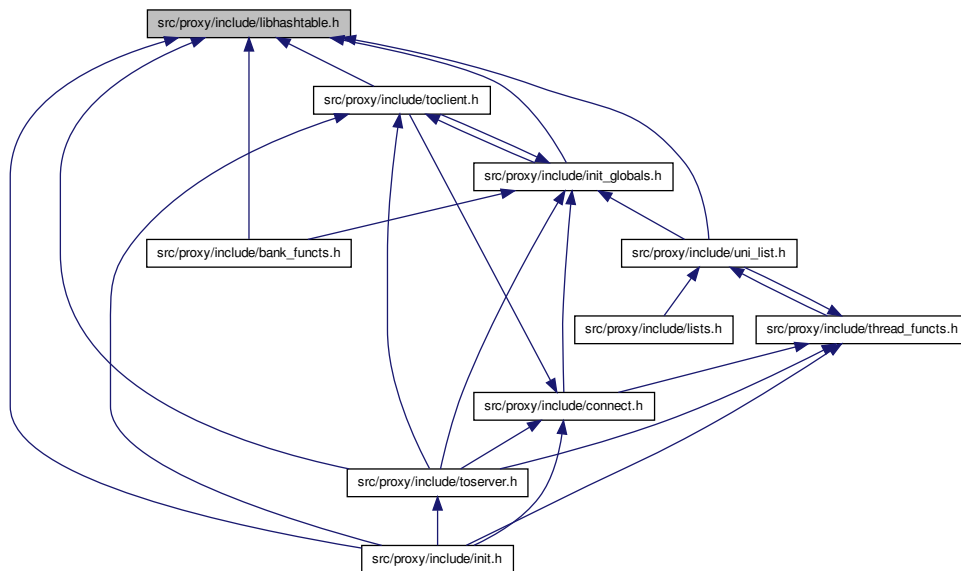
Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Grafo delle dipendenze di inclusione per libhashtable.h:



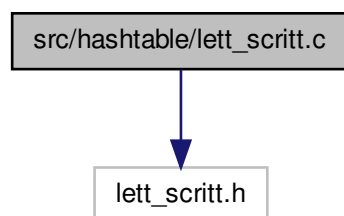
Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



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

<i>data</i> ,:	Indica la struttura dati sulla quale si effettua il lock
<i>external</i> ,:	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

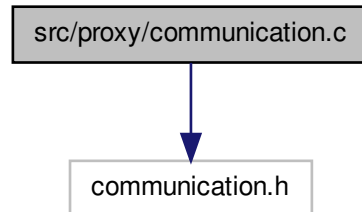
4.14.1.5 void `struct_end_write` (struct `CS_struct` * *data*)

`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 * [surfpo](#) (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

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

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

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

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

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

<i>text</i> ,:	Buffer di input
<i>buf</i> ,:	BUffer di output
<i>len</i> ,:	Lunghezza del buffer di output
<i>text_len</i> ,:	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

<i>URL</i> ,:	Contiene l'URL partendo da "mhttp://"
<i>dest</i> ,:	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 passati per parametri i valori assunti dallo header del messaggio.

Parametri

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

<i>len,:</i>	Lunghezza del parametro LEN della richiesta INF
<i>expire,:</i>	Expire time fornito dal messaggio dalla richiesta inf
<i>low,:</i>	Low range
<i>hight,:</i>	Hight Range
<i>cont,:</i>	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

<i>buf,:</i>	Buffer contenente la risorsa
<i>len,:</i>	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

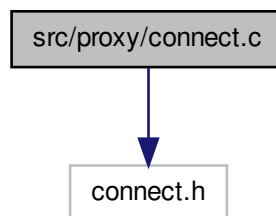
4.15.2 Documentazione delle variabili

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

<i>fd</i> ,:	Socket di comunicazione dal quale effettuare la lettura
<i>buf</i> ,:	Buffer all'interno del quale effettuare la scrittura
<i>len</i> ,:	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

<i>ptr</i> ,:	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'invio della richiesta tramite lettura in cache.

Parametri

<i>ptr,:</i>	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

<i>local,:</i>	Puntatore alla struttura dati di stato di connessione con il server
<i>resourcepath,:</i>	Stringa completa della risorsa da chiedere al server
<i>client,:</i>	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

<i>local,:</i>	Stato di connessione con il server
<i>res,:</i>	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

<i>ptr,:</i>	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

<i>fd,:</i>	File descriptor per l'invio
<i>buf,:</i>	Buffer contenente l'informazione da inviare
<i>len,:</i>	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

<i>ptr</i> ,:	Puntatore allo stato di connessione con il server
<i>buf</i> ,:	Buffer nel quale memorizzare la risposta
<i>len</i> ,:	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

<i>ptr</i> ,:	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

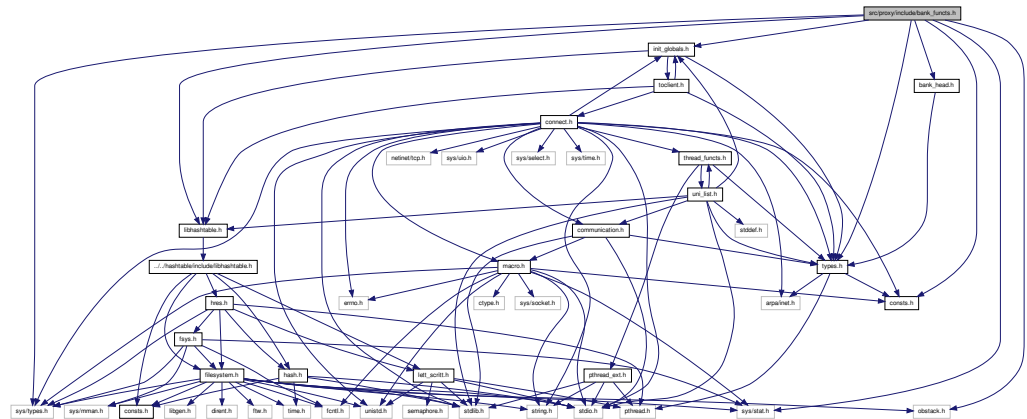
Parametri

<i>obtained_remote</i> ,:	Buffer all'interno del quale verrà memorizzata la risorsa
<i>resourcepath</i> ,:	Percorso remoto della risorsa da ottenere
<i>client</i> ,:	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"
```

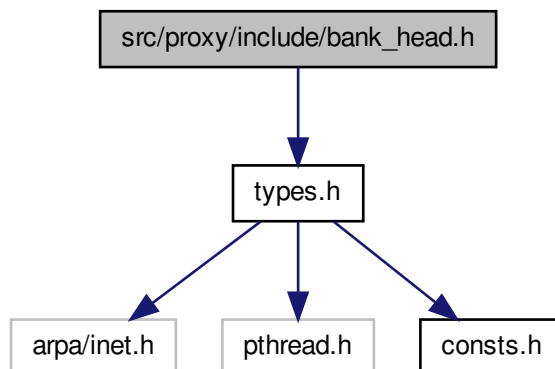
Grafo delle dipendenze di inclusione per bank_funcs.h:



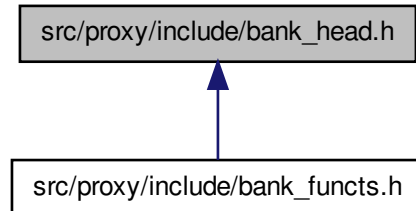
4.18 Riferimenti per il file src/proxy/include/bank_head.h

```
#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

<i>resource</i> ,:	risorsa che deve essere chiusa
<i>do_free_ - not_unmap</i> ,:	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

<i>remote_path,:</i>	Percorso della risorsa remota da richiedere al server
<i>dealloc,:</i>	Se è settato a vero, non viene settato il valore <i>local_path</i> , altrimenti viene aggiornato il suo valore
<i>local_path,:</i>	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 <i>dealloc</i> 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 existence_val, char * remote_path)

[cache_init\(\)](#): Funzione per la inizializzazione della struttura della risorsa per la cache.

Parametri

<i>Threads_Resource,:</i>	Risorsa locale associata al thread corrente. Può essere settata a NULL se <i>existence_val</i> == 2
<i>existence_val,:</i>	Valore di esistenza associato al file corrente (ottenuto da <i>cache_exists</i>)
<i>remote_path,:</i>	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, effettua la divisione nell'acquisizione della risorsa. Ritorna 0 in caso di successo.

Parametri

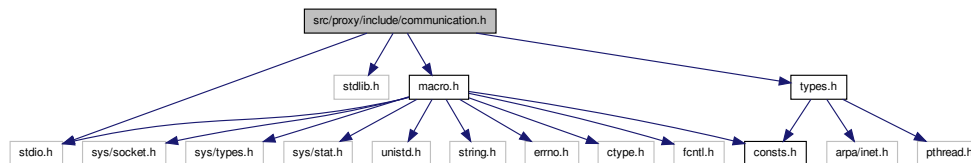
<i>Bart,:</i>	Struttura dati da aggiornare
<i>max_socks,:</i>	Numero totale dei sockets utilizzati dal thread corrente
<i>no,:</i>	Numero del socket corrente, contando da 1
<i>remote_file_size,:</i>	Dimensione del file remoto

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

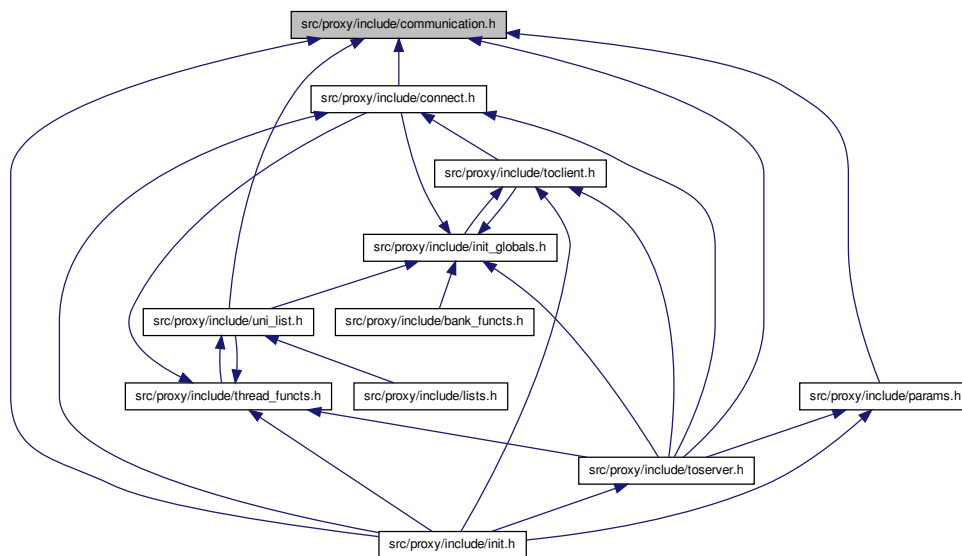
```
#include <stdio.h>
```

```
#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 high)
- int [doreply](#) (char *reply, int TYPE, int len, int empire, int low, int high, char *content)
- request * [parseRequest](#) (char *buf, int len)
- int [parseHead](#) (char *buf, int buflen, int *len, unsigned int *expire, int *low, int *high, 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 *high*, 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

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

4.19.2.2 int [dorequest](#) (char * *request*, int *TYPE*, char * *path*, int *low*, int *high*)

[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

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

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

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

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

<i>URL</i> ,:	Contiene l'URL partendo da "mhttp://"
<i>dest</i> ,:	Buffer 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 * high, 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 passati per parametri i valori assunti dallo header del messaggio.

Parametri

<i>buf,:</i>	Eventuale buffer dal quale leggere la risposta ricevuta
<i>buflen,:</i>	Lunghezza della parte di memoria effettiva in memoria
<i>len,:</i>	Lunghezza del parametro LEN della richiesta INF
<i>expire,:</i>	Expire time fornito dal messaggio dalla richiesta inf
<i>low,:</i>	Low range
<i>hight,:</i>	Hight Range
<i>cont,:</i>	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.

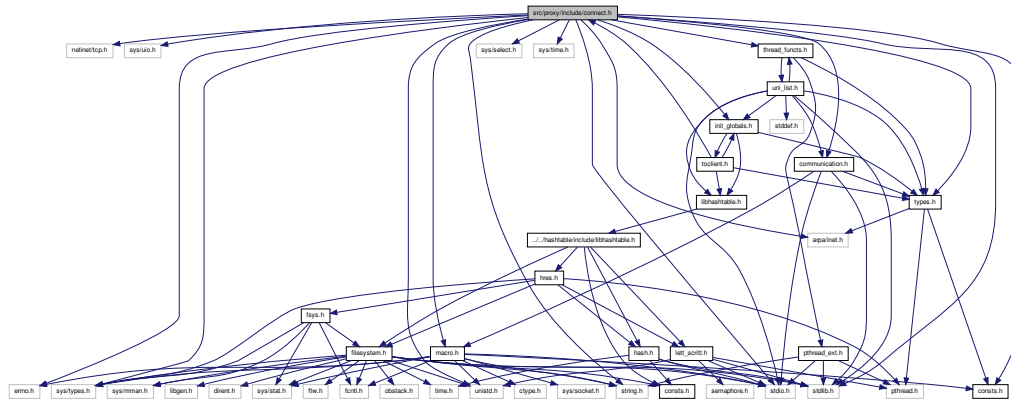
Parametri

<i>buf,:</i>	Buffer contenente la risorsa
<i>len,:</i>	Lunghezza della risorsa del buffer

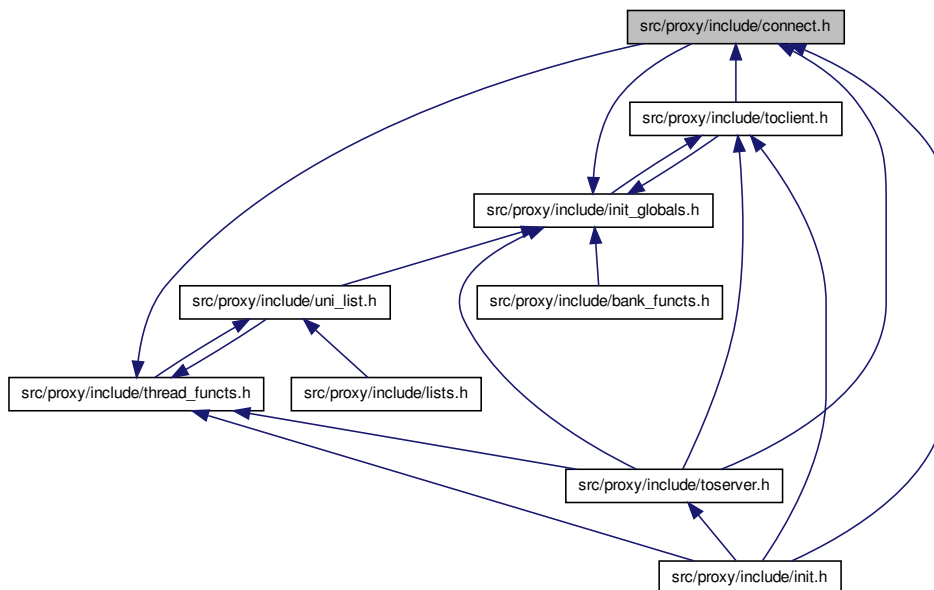
4.20 Riferimenti per il file `src/proxy/include/connect.h`

```
#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'invio della richiesta tramite lettura in cache.

Parametri

<i>ptr,:</i>	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

<i>local,:</i>	Stato di connessione con il server
<i>res,:</i>	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

<i>obtained_remote,:</i>	Buffer di memorizzazione della risposta
<i>resourcepath,:</i>	Percorso della risorsa remota
<i>client,:</i>	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

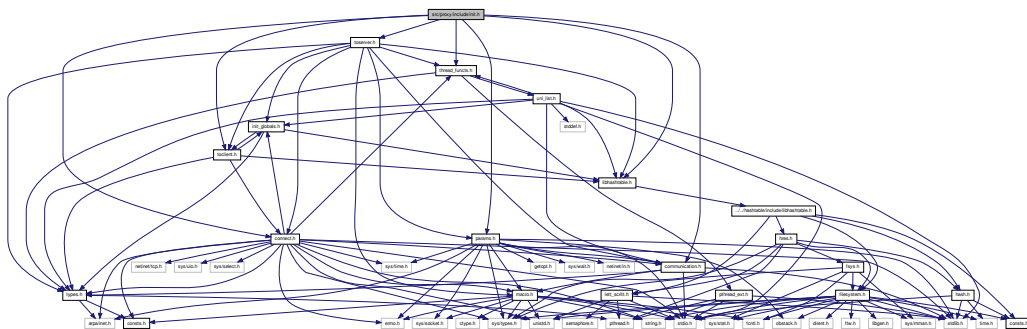
Parametri

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

<i>data</i> ,:	Informazione di stato del proxy
<i>fd_client</i> ,:	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

<i>data</i> ,:	Puntatore allo stato di connessione con il client
----------------	---

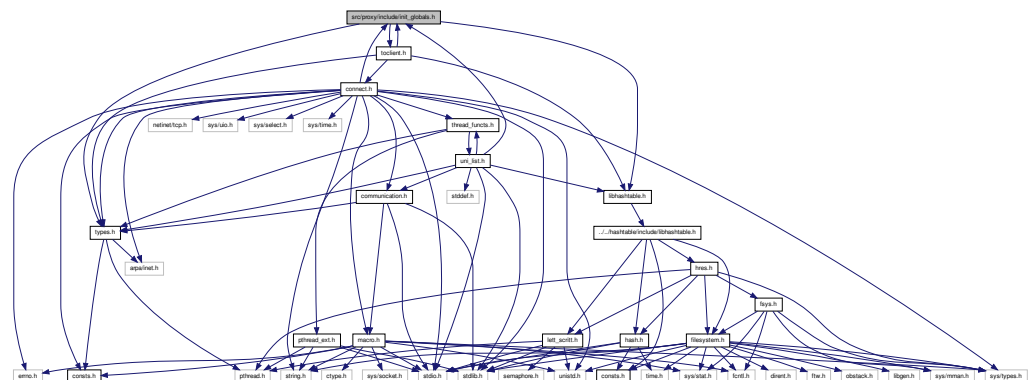
4.21.1.3 int start_threads ()

[start_threads\(\)](#): Funzione di inizializzazione dei threads e delle Pool

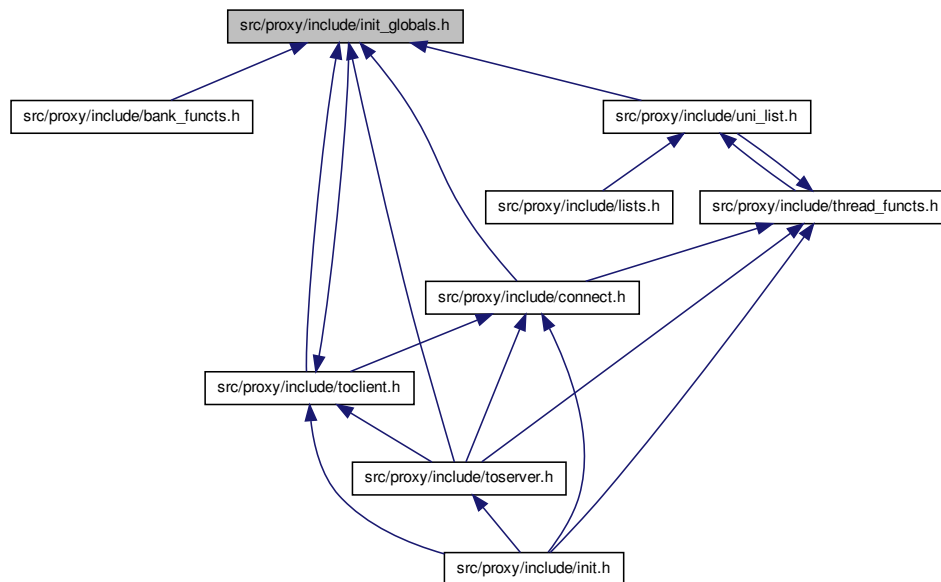
4.22 Riferimenti per il file src/proxy/include/init_globals.h

```
#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:

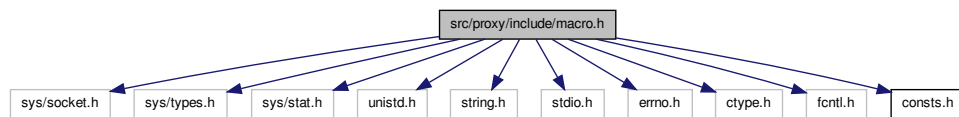


Variabili

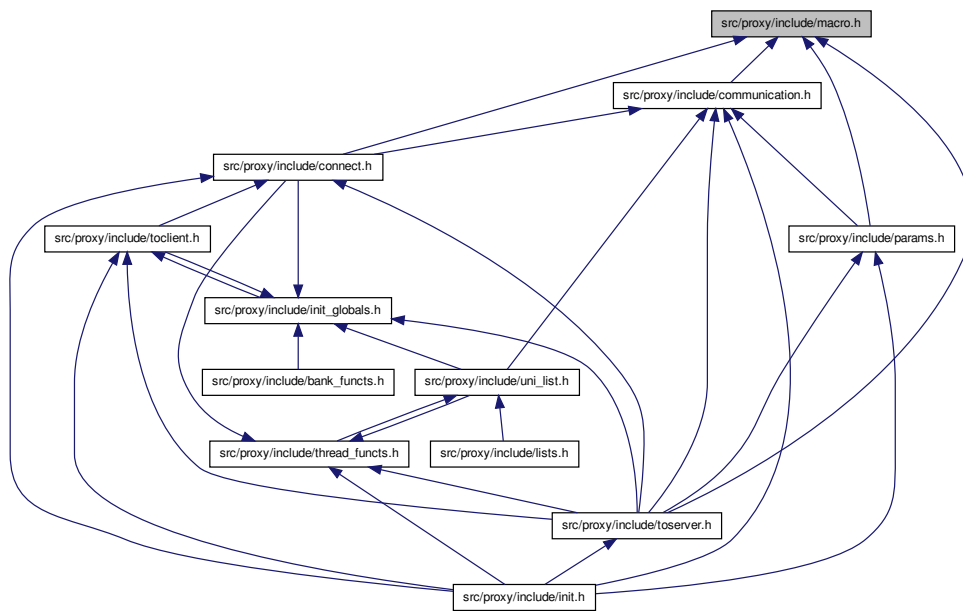
- `SCHEDULING serverPool`
- `SCHEDULING prefetchPool`
- `SCHEDULING cachePool`
- `struct proxy global`
- `ARGS_ONE prefetch_arg`
- `HRES cache_hash_table`


```
#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

<i>a,:</i>	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;
```

SET_TIMEOUT() Per automatizzare il settaggio del timeout

<i>timeout,</i> :	indica il tempo da resettare
-------------------	------------------------------

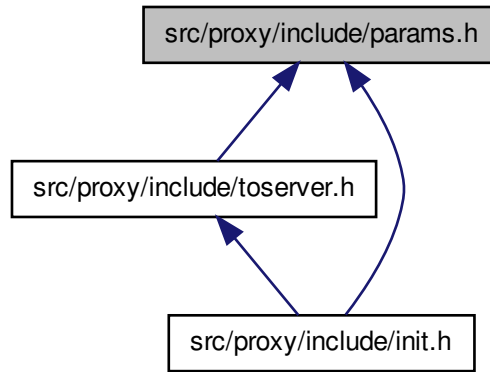
```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <errno.h>
#include <arpa/inet.h>
#include <obstack.h>
#include "macro.h"
#include "communication.h"
```

```

graph TD
    params["src/proxy/include/params.h"] --> communication["communication.h"]
    params --> getopt["getopt.h"]
    params --> wait["sys/wait.h"]
    params --> time["sys/time.h"]
    params --> inet["netinet/in.h"]
    params --> obstack["obstack.h"]
    
    communication --> macro["macro.h"]
    communication --> types["types.h"]
    communication --> stdlib["stdlib.h"]
    
    macro --> sys_types["sys/types.h"]
    macro --> sys_socket["sys/socket.h"]
    macro --> errno["errno.h"]
    macro --> stdio["stdio.h"]
    macro --> sys_stat["sys/stat.h"]
    macro --> unistd["unistd.h"]
    macro --> string["string.h"]
    macro --> ctype["ctype.h"]
    macro --> fcntl["fcntl.h"]
    macro --> consts["consts.h"]
    macro --> pthread["pthread.h"]
    macro --> arpa_inet["arpa/inet.h"]
    
    types --> string
    
    stdlib --> ctype

```

Generato il Fri Feb 17 2012 14:42:44 per Proxy(c)2012 da Doxygen



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 = mkshorptopt(longk);\
    ption_index,c,allocando) { \
        WHILE_GETOPT_L(argc,argv,longk,&o
        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 `mkshorptopt()`

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

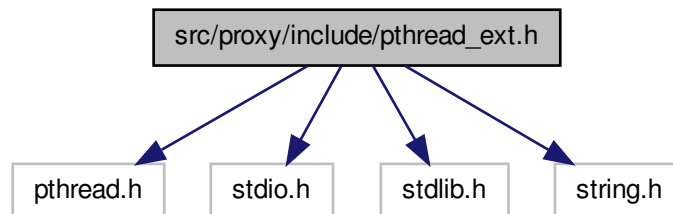
<i>args,:</i>	lunghezza dei parametri
<i>argv,:</i>	Parametri di lunghezza dichiarata nel parametro precedente
<i>params,:</i>	Puntatore allo stato di connessione lato client

INIZIO DEL RICONOSCIMENTO DEI PARAMETRI

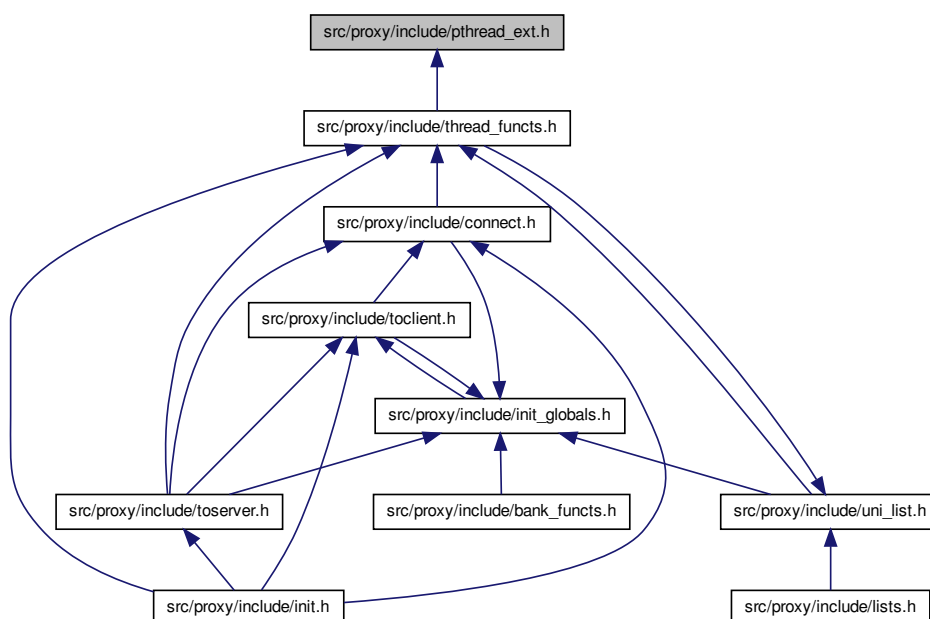
4.26 Riferimenti per il file src/proxy/include/pthread_ext.h

```
#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:

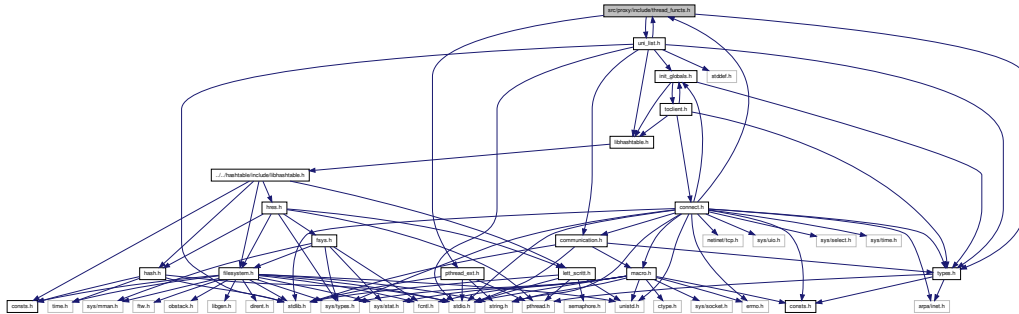
```
do {
    int rtn = pthread_mutex_init(x, NULL); \
    if (rtn) { \
        fprintf(stderr, "Error while init mutex: %s\n", (c \
        har*)strerror(rtn)); \
        exit(1); \
    } \
} while (0)
```

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

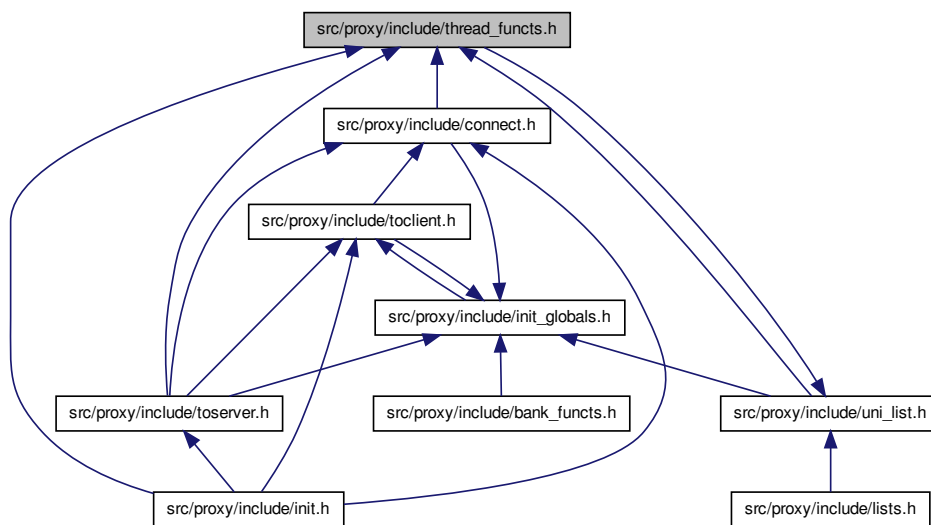
4.27 Riferimenti per il file `src/proxy/include/thread_funcs.h`

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

Grafo delle dipendenze di inclusione per thread_funcs.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 prioritize)

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

<i>timbracartellino,</i> :	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

<i>todo,</i> :	Puntatore allo scheduler
<i>func,</i> :	Puntatore alla funzione da eseguire
<i>args,</i> :	Puntatore agli argomenti della lista
<i>priorize,</i> :	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

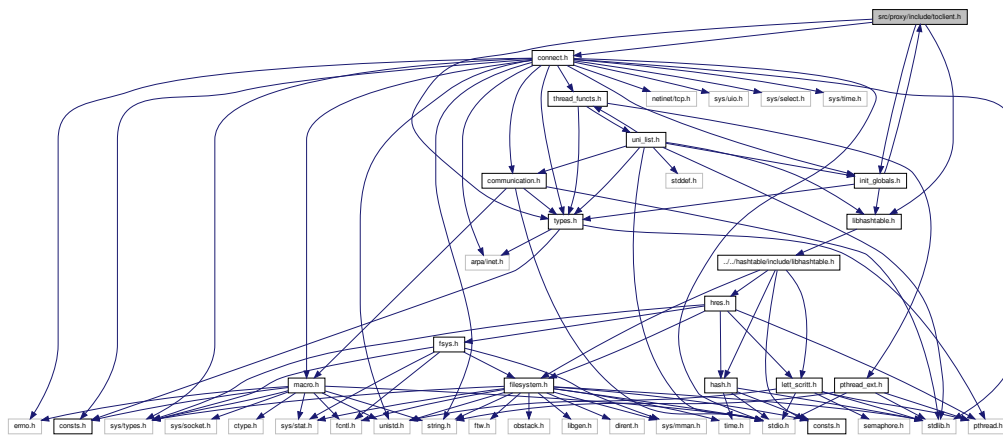
Parametri

<i>x,</i> :	parametro di tipo ARGS_ONE
-------------	--

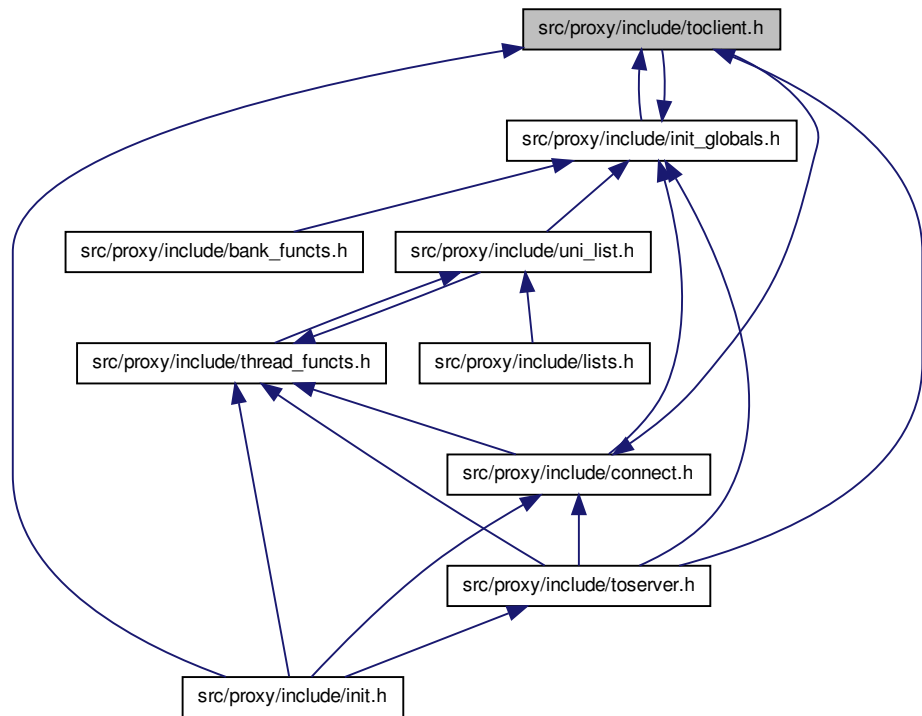
4.28 Riferimenti per il file src/proxy/include/toclient.h

```
#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

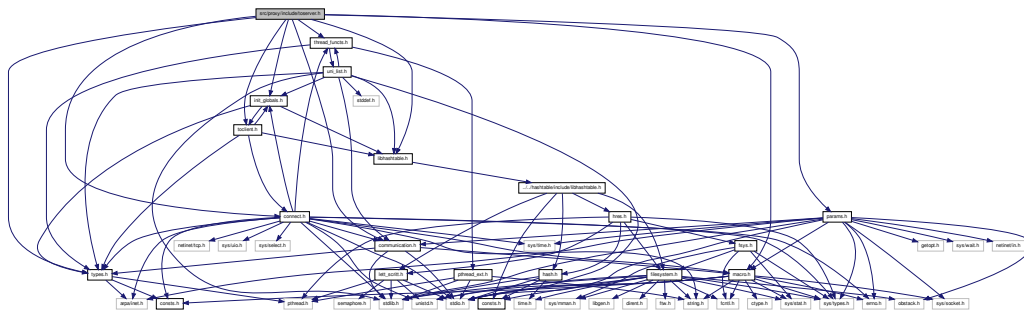
4.28.1.1 void toclient (int *whoami*, void * *arg*)

[toclient\(\)](#): Inoltra la risposta al client, tramite lettura della cache

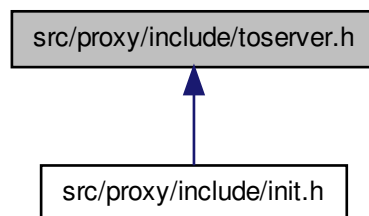
4.29 Riferimenti per il file src/proxy/include/toserver.h

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

<i>buffer,:</i>	Area di memoria dove verrà salvata la risorsa
<i>remote_path,:</i>	Percorso remoto associato alla risorsa
<i>fdclient,:</i>	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

<i>whoami,:</i>	Indicatore del thread corrente
<i>prefetch_res:Indicazion</i>	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

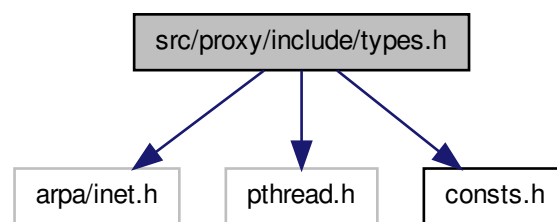
Parametri

<i>whoami,:</i>	Indicatore del thread corrente
<i>params,:</i>	Parametri indicanti la richiesta

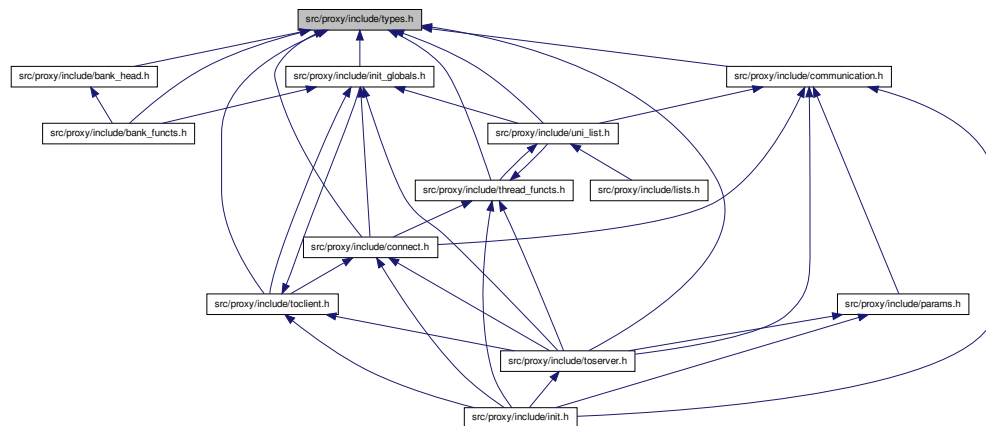
4.30 Riferimenti per il file src/proxy/include/types.h

```
#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](#)

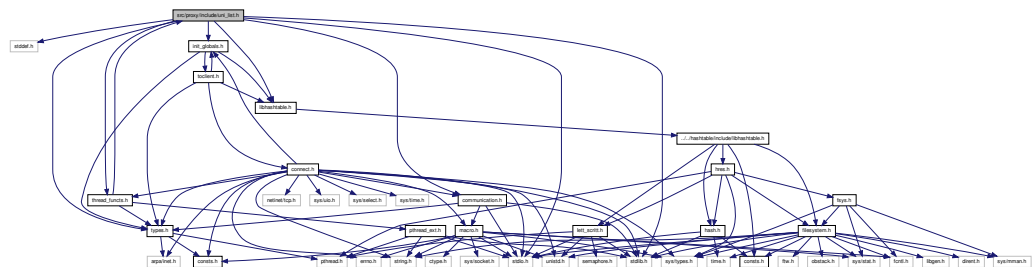
4.30.1 Documentazione delle ridefinizioni di tipo (typedef)

4.30.1.1 typedef struct [__mialista__](#) [LIST](#)

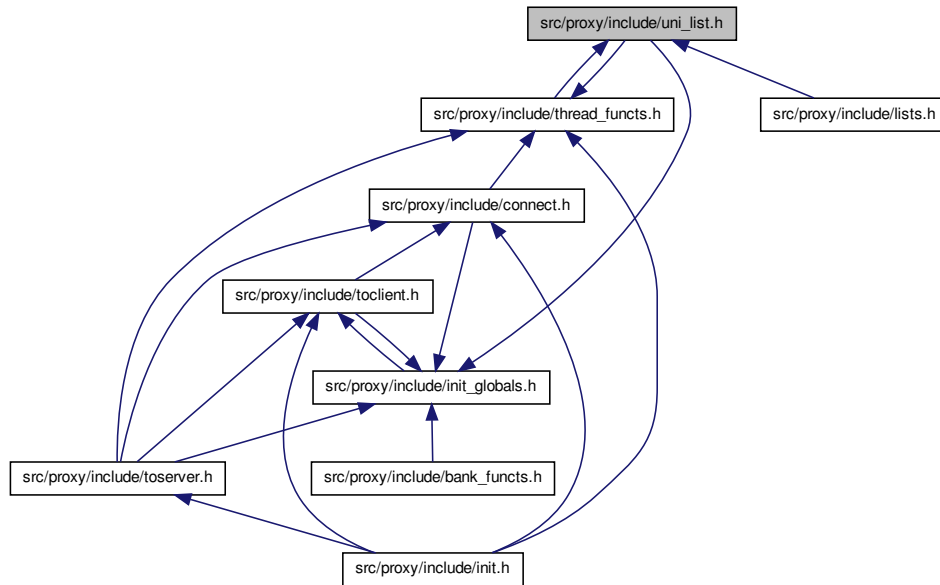
4.31 Riferimenti per il file `src/proxy/include/uni_list.h`

```
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include "libhashtable.h"
#include "thread_funcs.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 bufferhigh)`

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 bufferhigh)**

[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.

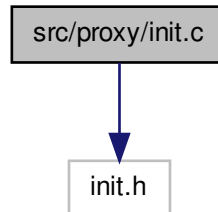
Parametri

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

<i>data</i> ,:	Informazione di stato del proxy
<i>fd_client</i> ,:	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

<i>data</i> ,:	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]`

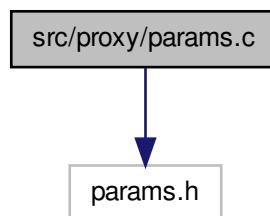
4.32.2.11 `ARGS_ONE` `server_thread_args[4]`

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 * [mkshoropt](#) (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

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

INIZIO DEL RICONOSCIMENTO DEI PARAMETRI

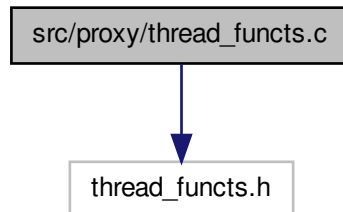
4.33.1.3 char * [mkshoropt](#) (struct option * *longopt*)

[mkshoropt\(\)](#): 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_funcs.c

```
#include "thread_funcs.h"
```

Grafo delle dipendenze di inclusione per thread_funcs.c:



Funzioni

- int `init_scheduler` (`SCHEDULING` *timbracartellino)
- void * `thread_memento` (void *x)
- void `main_add_job` (`SCHEDULING` *todo, void(*func)(), void *args, char prioritize)

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

<i>timbracartellino</i> ,:	Puntatore alla Pool associata.
----------------------------	--------------------------------

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

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

Parametri

<i>todo</i> ,:	Puntatore allo scheduler
<i>func</i> ,:	Puntatore alla funzione da eseguire
<i>args</i> ,:	Puntatore agli argomenti della lista
<i>priorize</i> ,:	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

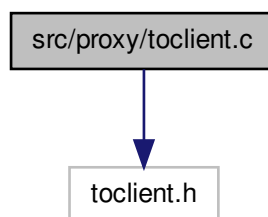
Parametri

<i>x</i> ,:	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

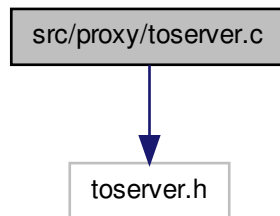
4.35.1.1 void toclient (int *whoami*, void * *arg*)

[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

<i>whoami</i> ,:	Indicatore del thread corrente
------------------	--------------------------------

<i>prefetch_res</i> :	Indicazione della risorsa da ottenere
-----------------------	---------------------------------------

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

[handle_serverPool\(\)](#): Gestione della richiesta client

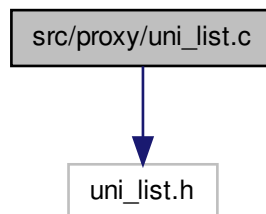
Parametri

<i>whoami</i> :	Indicatore del thread corrente
<i>params</i> :	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 `bufferhigh`)

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 bufferhigh)**

[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

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

4.37.2 Documentazione delle variabili**4.37.2.1 int oldoutptfd**

Indice analitico

- `_BSD_SOURCE`
 - `hash.h`, [43](#)
 - `_GNU_SOURCE`
 - `communication.h`, [64](#)
 - `filesystem.h`, [36](#)
 - `__mialista__`, [5](#)
 - `next`, [5](#)
- `alloc_new_job`
 - `uni_list.c`, [100](#)
 - `uni_list.h`, [90](#)
- `ANYTYPE`
 - `lett_scritt.h`, [48](#)
- `args`
 - `JOB`, [11](#)
- `ARGS_ONE`, [6](#)
 - `sched`, [7](#)
 - `whoami`, [7](#)
- `atreturn`
 - `communication.c`, [53](#)
- `atreturns`
 - `communication.c`, [53](#)
- `attr`
 - `CS_struct`, [9](#)
- `bank_head.h`
 - `cache_done`, [61](#)
 - `cache_exists`, [61](#)
 - `cache_init`, [62](#)
 - `cache_init_socket`, [62](#)
- `base_read`
 - `connect.c`, [57](#)
- `BLACK`
 - `proxy/include/consts.h`, [33](#)
- `BLUE`
 - `proxy/include/consts.h`, [33](#)
- `BOLD`
 - `proxy/include/consts.h`, [33](#)
- `cache`
 - `init.c`, [94](#)
- `cache_arg`
 - `init.c`, [94](#)
- `cache_done`
 - `bank_head.h`, [61](#)
- `cache_exists`
 - `bank_head.h`, [61](#)
- `cache_free`
 - `filesystem.h`, [36](#)
- `cache_hash_table`
 - `init.c`, [94](#)
 - `init_globals.h`, [72](#)
- `cache_init`
 - `bank_head.h`, [62](#)
- `cache_init_socket`
 - `bank_head.h`, [62](#)
- `cachePool`
 - `init.c`, [94](#)
 - `init_globals.h`, [72](#)
- `close_cached_file`
 - `hres.c`, [27](#)
 - `hres.h`, [45](#)
- `close_map`
 - `filesystem.h`, [36](#)
- `communication.c`
 - `atreturn`, [53](#)
 - `atreturns`, [53](#)
 - `doreply`, [54](#)
 - `dorequest`, [54](#)
 - `get_resource_alloc`, [54](#)
 - `getfirstlevel_next`, [55](#)
 - `getIP`, [55](#)
 - `getPort`, [55](#)
 - `haspos`, [55](#)
 - `oldoutptfd`, [56](#)
 - `parseHead`, [55](#)
 - `parseRequest`, [56](#)
 - `surfto`, [56](#)
- `communication.h`
 - `_GNU_SOURCE`, [64](#)
 - `distance`, [64](#)
 - `doreply`, [64](#)

- dorequest, 64
- get_resource_alloc, 65
- getIP, 65
- getPort, 65
- parseHead, 65
- parseRequest, 66
- cond_init
 - pthread_ext.h, 80
- connect.c
 - base_read, 57
 - establishConnection, 57
 - fallisci_tutto, 57
 - INIT_CONNECT, 57
 - init_connect_struct, 58
 - invia_client_termina, 58
 - send_request, 58
 - server_base_send, 58
 - server_complete_read, 58
 - server_complete_send, 59
 - session, 59
- connect.h
 - fallisci_tutto, 68
 - invia_client_termina, 68
 - MAX_TENTATIVI, 68
 - session, 68
- connection, 7
 - expiry, 8
 - fd_client, 8
 - fd_server, 8
 - PATH, 8
 - sIP, 8
 - sPort, 8
 - tentativi, 8
- container_of2
 - uni_list.h, 90
- CPrintf
 - macro.h, 74
- CS_struct, 8
 - attr, 9
- CYAN
 - proxy/include/consts.h, 33
- dequeue
 - uni_list.c, 100
 - uni_list.h, 90
- dequeue_job
 - uni_list.c, 100
 - uni_list.h, 90
- distance
 - communication.h, 64
- doreply
 - communication.c, 54
 - communication.h, 64
- dorequest
 - communication.c, 54
 - communication.h, 64
- elem
 - list_pprint, 14
- elemen
 - request_cache, 17
- elements
 - hash_tbl, 9
- END_COLOR
 - proxy/include/consts.h, 33
- END_CYCLE
 - params.h, 77
- enqueue
 - uni_list.c, 100
 - uni_list.h, 90
- enqueue_job
 - uni_list.c, 100
 - uni_list.h, 90
- establishConnection
 - connect.c, 57
- exists_job_res
 - uni_list.c, 100
 - uni_list.h, 90
- EXPIRED
 - filesystem.h, 36
- expiry
 - connection, 8
- fallisci_tutto
 - connect.c, 57
 - connect.h, 68
- fd
 - proxy, 16
- fd_client
 - connection, 8
 - request_cache, 17
- fd_file
 - request_cache, 17
- fd_server
 - connection, 8
- fetch
 - toserver.c, 98
 - toserver.h, 86
- file_exists
 - filesystem.c, 22

- filesystem.h, 36
- FILE_SIZE
 - filesystem.h, 36
- FileClose
 - fsys.c, 24
 - fsys.h, 40
- FileCreate
 - fsys.c, 24
 - fsys.h, 40
- FileOpen
 - fsys.c, 25
 - fsys.h, 40
- filesystem.c
 - file_exists, 22
 - folder_empty, 22
 - obtain_local, 22
 - re_new_resource, 22
 - recursiveDelete, 22
 - resource_exists, 23
 - resource_remove, 23
- filesystem.h
 - _GNU_SOURCE, 36
 - cache_free, 36
 - close_map, 36
 - EXPIRED, 36
 - file_exists, 36
 - FILE_SIZE, 36
 - folder_empty, 37
 - NOW, 36
 - obtain_local, 37
 - PAST, 36
 - re_new_resource, 37
 - recursiveDelete, 37
 - resource_exists, 37
 - resource_remove, 38
- FINEDIMONDO
 - thread_funcs.h, 82
- folder_empty
 - filesystem.c, 22
 - filesystem.h, 37
- fsys.c
 - FileClose, 24
 - FileCreate, 24
 - FileOpen, 25
- fsys.h
 - FileClose, 40
 - FileCreate, 40
 - FileOpen, 40
- GET
 - proxy/include/consts.h, 33
- GET_NEXT_OPTIND
 - params.h, 77
- get_resource_alloc
 - communication.c, 54
 - communication.h, 65
- getfirstlevel_next
 - communication.c, 55
- getIP
 - communication.c, 55
 - communication.h, 65
- GETOPT_L
 - params.h, 77
- getPort
 - communication.c, 55
 - communication.h, 65
- global
 - init.c, 94
 - init_globals.h, 72
- GREEN
 - proxy/include/consts.h, 33
- handle_client_request
 - init.c, 92
 - init.h, 69
- handle_file_filesystem
 - hres.c, 27
 - hres.h, 45
- handle_prefetchPool
 - toserver.c, 98
 - toserver.h, 86
- handle_serverPool
 - toserver.c, 99
 - toserver.h, 86
- HAS_ARG
 - params.h, 77
- HAS_OTHER_ARGS
 - params.h, 77
- hash
 - hash.c, 26
 - hash.h, 43
- hash.c
 - hash, 26
 - hash_exists, 26
 - hash_insert, 26
 - hash_occupied, 26
 - hash_remove, 26
 - hashb, 26
 - init_hashtbl, 26
 - SEED, 26

- hash.h
 - _BSD_SOURCE, 43
 - hash, 43
 - hash_exists, 43
 - hash_insert, 43
 - hash_occupied, 43
 - hash_remove, 43
 - hashb, 43
 - init_hashtbl, 43
- hash_exists
 - hash.c, 26
 - hash.h, 43
- hash_insert
 - hash.c, 26
 - hash.h, 43
- hash_occupied
 - hash.c, 26
 - hash.h, 43
- hash_remove
 - hash.c, 26
 - hash.h, 43
- HASH_SIZE
 - hashtable/include/consts.h, 30
- hash_tbl, 9
 - elements, 9
 - value, 9
- hashb
 - hash.c, 26
 - hash.h, 43
- hashtable
 - HRES, 10
- hashtable/include/consts.h
 - HASH_SIZE, 30
 - MAX_FILE, 30
 - MAX_HRES, 30
 - MOD, 30
 - MULT1, 30
 - MULT2, 30
 - NHASH1, 30
 - NHASH2, 30
- haspos
 - communication.c, 55
- head
 - JOB_QUEUE, 12
- help
 - params.c, 95
- HRES, 9
 - hashtable, 10
 - lettscrit, 10
- hres.c
 - close_cached_file, 27
 - handle_file_filesystem, 27
 - init_hres, 28
 - is_inhash, 28
 - map_cached_file, 28
 - parseHead_time, 28
- hres.h
 - close_cached_file, 45
 - handle_file_filesystem, 45
 - init_hres, 45
 - is_inhash, 45
 - map_cached_file, 45
- INF
 - proxy/include/consts.h, 33
- INFO
 - proxy/include/consts.h, 33
- init.c
 - cache, 94
 - cache_arg, 94
 - cache_hash_table, 94
 - cachePool, 94
 - global, 94
 - handle_client_request, 92
 - init_proxy_clientside, 93
 - main, 93
 - prefetch, 94
 - prefetch_arg, 94
 - prefetchPool, 94
 - pt_attr, 94
 - server, 94
 - server_thread_args, 94
 - serverPool, 94
 - start_threads, 93
- init.h
 - handle_client_request, 69
 - init_proxy_clientside, 70
 - start_threads, 70
- INIT_CONNECT
 - connect.c, 57
- init_connect_struct
 - connect.c, 58
- init_globals.h
 - cache_hash_table, 72
 - cachePool, 72
 - global, 72
 - prefetch_arg, 72
 - prefetchPool, 72
 - serverPool, 72
- init_hashtbl

- hash.c, [26](#)
- hash.h, [43](#)
- init_hres
 - hres.c, [28](#)
 - hres.h, [45](#)
- init_job_queue
 - uni_list.c, [100](#)
 - uni_list.h, [90](#)
- init_list_head
 - uni_list.h, [90](#)
- init_params
 - params.c, [95](#)
 - params.h, [78](#)
- init_proxy_clientside
 - init.c, [93](#)
 - init.h, [70](#)
- init_scheduler
 - thread_funcs.c, [96](#)
 - thread_funcs.h, [82](#)
- init_struct
 - lett_scritt.c, [52](#)
 - lett_scritt.h, [48](#)
- INTERVAL_NOT_FOUND
 - proxy/include/consts.h, [33](#)
- invia_client_termina
 - connect.c, [58](#)
 - connect.h, [68](#)
- ip
 - proxy, [16](#)
- IS_EOOPT
 - params.h, [77](#)
- is_inhash
 - hres.c, [28](#)
 - hres.h, [45](#)
- JOB, [11](#)
 - args, [11](#)
 - procedura, [11](#)
- JOB_QUEUE, [11](#)
 - head, [12](#)
 - todo, [12](#)
- kassert
 - macro.h, [74](#)
- lett_scritt.c
 - init_struct, [52](#)
 - struct_do_read, [52](#)
 - struct_do_write, [52](#)
 - struct_end_read, [52](#)
 - struct_end_write, [52](#)
- lett_scritt.h
 - ANYTYPE, [48](#)
 - init_struct, [48](#)
 - P, [48](#)
 - struct_do_read, [48](#)
 - struct_do_write, [48](#)
 - struct_end_read, [48](#)
 - struct_end_write, [49](#)
 - V, [48](#)
 - WREAD, [48](#)
 - WWRITE, [48](#)
- lettscriit
 - HRES, [10](#)
- level
 - proxy, [16](#)
- LIMIT_FILE
 - proxy/include/consts.h, [33](#)
- LIST
 - types.h, [88](#)
- list_print, [13](#)
 - elem, [14](#)
 - valore, [14](#)
- LONGNAME
 - params.h, [77](#)
- macro.h
 - CPrintf, [74](#)
 - kassert, [74](#)
 - ONCONDRET1, [74](#)
 - SET_TIMEOUT, [74](#)
- MAGENTA
 - proxy/include/consts.h, [33](#)
- main
 - init.c, [93](#)
- main_add_job
 - thread_funcs.c, [96](#)
 - thread_funcs.h, [82](#)
- map_cached_file
 - hres.c, [28](#)
 - hres.h, [45](#)
- MAX_BUFF
 - proxy/include/consts.h, [33](#)
- MAX_FILE
 - hashtable/include/consts.h, [30](#)
 - proxy/include/consts.h, [33](#)
- MAX_HRES
 - hashtable/include/consts.h, [30](#)
- MAX_PATH
 - proxy/include/consts.h, [33](#)

- MAX_TENTATIVI
 - connect.h, [68](#)
- MAX_TIME
 - proxy/include/consts.h, [33](#)
- MAX_TIMESELECT
 - proxy/include/consts.h, [33](#)
- MAX_uTIME
 - proxy/include/consts.h, [33](#)
- MAXCONN
 - proxy/include/consts.h, [33](#)
- MAXIDREF
 - proxy/include/consts.h, [33](#)
- MAXREF
 - proxy/include/consts.h, [33](#)
- MAXREQUEST
 - proxy/include/consts.h, [33](#)
- meStesso
 - proxy, [16](#)
- mkshortopt
 - params.c, [95](#)
- MOD
 - hashtable/include/consts.h, [30](#)
- msg, [14](#)
 - rq, [15](#)
 - sock, [15](#)
- MULT1
 - hashtable/include/consts.h, [30](#)
- MULT2
 - hashtable/include/consts.h, [30](#)
- mutex_init
 - pthread_ext.h, [80](#)
- new
 - uni_list.c, [100](#)
- new_list
 - uni_list.h, [91](#)
- next
 - __mialista__, [5](#)
- NHASH1
 - hashtable/include/consts.h, [30](#)
- NHASH2
 - hashtable/include/consts.h, [30](#)
- NO_MORE_THREAD
 - proxy/include/consts.h, [33](#)
- non_empty
 - SCHEDULING, [20](#)
- NOT_FOUND
 - proxy/include/consts.h, [33](#)
- NOW
 - filesystem.h, [36](#)
- obstack_chunk_alloc
 - params.h, [77](#)
- obstack_chunk_free
 - params.h, [77](#)
- obtain_local
 - filesystem.c, [22](#)
 - filesystem.h, [37](#)
- OK
 - proxy/include/consts.h, [33](#)
- OK_INFO
 - proxy/include/consts.h, [33](#)
- OK_RANGE
 - proxy/include/consts.h, [33](#)
- oldoutptfd
 - communication.c, [56](#)
 - thread_funcs.c, [97](#)
 - uni_list.c, [101](#)
- ONCONDRETI
 - macro.h, [74](#)
- P
 - lett_scritt.h, [48](#)
- params.c
 - help, [95](#)
 - init_params, [95](#)
 - mkshortopt, [95](#)
- params.h
 - END_CYCLE, [77](#)
 - GET_NEXT_OPTIND, [77](#)
 - GETOPT_L, [77](#)
 - HAS_ARG, [77](#)
 - HAS_OTHER_ARGS, [77](#)
 - init_params, [78](#)
 - IS_EOOPT, [77](#)
 - LONGNAME, [77](#)
 - obstack_chunk_alloc, [77](#)
 - obstack_chunk_free, [77](#)
 - REMAINING_ARGS, [77](#)
 - START_CYCLE, [77](#)
 - WHILE_GETOPT_L, [78](#)
- parseHead
 - communication.c, [55](#)
 - communication.h, [65](#)
- parseHead_time
 - hres.c, [28](#)
- parseRequest
 - communication.c, [56](#)
 - communication.h, [66](#)
- PAST
 - filesystem.h, [36](#)

- PATH
 - connection, 8
 - request, 16
- planning
 - SCHEDULING, 20
- port
 - proxy, 16
- prefetch
 - init.c, 94
- prefetch_arg
 - init.c, 94
 - init_globals.h, 72
- prefetch_num
 - reslist, 18
- prefetchPool
 - init.c, 94
 - init_globals.h, 72
- procedura
 - JOB, 11
- proxy, 15
 - fd, 16
 - ip, 16
 - level, 16
 - meStesso, 16
 - port, 16
- proxy/include/consts.h
 - BLACK, 33
 - BLUE, 33
 - BOLD, 33
 - CYAN, 33
 - END_COLOR, 33
 - GET, 33
 - GREEN, 33
 - INF, 33
 - INFO, 33
 - INTERVAL_NOT_FOUND, 33
 - LIMIT_FILE, 33
 - MAGENTA, 33
 - MAX_BUFF, 33
 - MAX_FILE, 33
 - MAX_PATH, 33
 - MAX_TIME, 33
 - MAX_TIMESELECT, 33
 - MAX_uTIME, 33
 - MAXCONN, 33
 - MAXIDREF, 33
 - MAXREF, 33
 - MAXREQUEST, 33
 - NO_MORE_THREAD, 33
 - NOT_FOUND, 33
 - OK, 33
 - OK_INFO, 33
 - OK_RANGE, 33
 - PROXY_IP, 33
 - PROXY_PORT, 33
 - RED, 33
 - RESET_COLOR, 33
 - SOCKET_ERROR, 33
 - START_COLOR, 33
 - UNKNOWN_ERROR, 33
 - UNKNOWN_ERROR_SIZE, 33
 - WHITE, 33
 - WRONG_REQUEST, 33
 - YELLOW, 33
- PROXY_IP
 - proxy/include/consts.h, 33
- PROXY_PORT
 - proxy/include/consts.h, 33
- pt_attr
 - init.c, 94
- pthread_ext.h
 - cond_init, 80
 - mutex_init, 80
- push
 - uni_list.c, 101
- push_job
 - uni_list.c, 101
 - uni_list.h, 91
- qlock
 - SCHEDULING, 20
- qsize
 - SCHEDULING, 20
- rangehigh
 - request, 16
- rangelow
 - request, 17
- re_new_resource
 - filesystem.c, 22
 - filesystem.h, 37
- recursiveDelete
 - filesystem.c, 22
 - filesystem.h, 37
- RED
 - proxy/include/consts.h, 33
- REMAINING_ARGS
 - params.h, 77
- remote
 - request_cache, 17

- remote_path
 - reslist, [18](#)
- remove_elem
 - uni_list.c, [101](#)
 - uni_list.h, [91](#)
- remove_job
 - uni_list.c, [101](#)
 - uni_list.h, [91](#)
- request, [16](#)
 - PATH, [16](#)
 - rangehigh, [16](#)
 - rangelow, [17](#)
 - TYPE, [17](#)
- request_cache, [17](#)
 - elemen, [17](#)
 - fd_client, [17](#)
 - fd_file, [17](#)
 - remote, [17](#)
- RESET_COLOR
 - proxy/include/consts.h, [33](#)
- reslist, [17](#)
 - prefetch_num, [18](#)
 - remote_path, [18](#)
- resource_exists
 - filesystem.c, [23](#)
 - filesystem.h, [37](#)
- resource_remove
 - filesystem.c, [23](#)
 - filesystem.h, [38](#)
- rq
 - msg, [15](#)
- run_job
 - uni_list.c, [101](#)
 - uni_list.h, [91](#)
- sched
 - ARGS_ONE, [7](#)
- SCHEDULING, [18](#)
 - non_empty, [20](#)
 - planning, [20](#)
 - qlock, [20](#)
 - qsize, [20](#)
- SEED
 - hash.c, [26](#)
- send_request
 - connect.c, [58](#)
- server
 - init.c, [94](#)
- server_base_send
 - connect.c, [58](#)
- server_complete_read
 - connect.c, [58](#)
- server_complete_send
 - connect.c, [59](#)
- server_thread_args
 - init.c, [94](#)
- serverPool
 - init.c, [94](#)
 - init_globals.h, [72](#)
- session
 - connect.c, [59](#)
 - connect.h, [68](#)
- SET_TIMEOUT
 - macro.h, [74](#)
- sIP
 - connection, [8](#)
- sock
 - msg, [15](#)
- SOCKET_ERROR
 - proxy/include/consts.h, [33](#)
- sPort
 - connection, [8](#)
- src/hashtable/filesystem.c, [21](#)
- src/hashtable/fsys.c, [23](#)
- src/hashtable/hash.c, [25](#)
- src/hashtable/hres.c, [27](#)
- src/hashtable/include/consts.h, [29](#)
- src/hashtable/include/filesystem.h, [33](#)
- src/hashtable/include/fsys.h, [38](#)
- src/hashtable/include/hash.h, [41](#)
- src/hashtable/include/hres.h, [43](#)
- src/hashtable/include/lett_scritt.h, [46](#)
- src/hashtable/include/libhashtable.h, [49](#)
- src/hashtable/lett_scritt.c, [51](#)
- src/proxy/communication.c, [52](#)
- src/proxy/connect.c, [56](#)
- src/proxy/include/bank_funcs.h, [59](#)
- src/proxy/include/bank_head.h, [60](#)
- src/proxy/include/communication.h, [62](#)
- src/proxy/include/connect.h, [66](#)
- src/proxy/include/consts.h, [30](#)
- src/proxy/include/init.h, [69](#)
- src/proxy/include/init_globals.h, [70](#)
- src/proxy/include/libhashtable.h, [50](#)
- src/proxy/include/lists.h, [72](#)
- src/proxy/include/macro.h, [72](#)
- src/proxy/include/params.h, [75](#)
- src/proxy/include/pthread_ext.h, [78](#)
- src/proxy/include/thread_funcs.h, [80](#)
- src/proxy/include/toclient.h, [83](#)

- src/proxy/include/toserver.h, 85
- src/proxy/include/types.h, 87
- src/proxy/include/uni_list.h, 88
- src/proxy/init.c, 91
- src/proxy/params.c, 94
- src/proxy/thread_funcs.c, 95
- src/proxy/toclient.c, 97
- src/proxy/toserver.c, 98
- src/proxy/uni_list.c, 99
- START_COLOR
 - proxy/include/consts.h, 33
- START_CYCLE
 - params.h, 77
- start_threads
 - init.c, 93
 - init.h, 70
- struct_do_read
 - lett_scritt.c, 52
 - lett_scritt.h, 48
- struct_do_write
 - lett_scritt.c, 52
 - lett_scritt.h, 48
- struct_end_read
 - lett_scritt.c, 52
 - lett_scritt.h, 48
- struct_end_write
 - lett_scritt.c, 52
 - lett_scritt.h, 49
- surfto
 - communication.c, 56
- tentativi
 - connection, 8
- thread_funcs.c
 - init_scheduler, 96
 - main_add_job, 96
 - oldoutptfd, 97
 - thread_memento, 97
- thread_funcs.h
 - FINEDIMONDO, 82
 - init_scheduler, 82
 - main_add_job, 82
 - thread_memento, 82
- thread_memento
 - thread_funcs.c, 97
 - thread_funcs.h, 82
- toclient
 - toclient.c, 98
 - toclient.h, 84
- toclient.c
 - toclient, 98
- toclient.h
 - toclient, 84
- todo
 - JOB_QUEUE, 12
- toserver.c
 - fetch, 98
 - handle_prefetchPool, 98
 - handle_serverPool, 99
- toserver.h
 - fetch, 86
 - handle_prefetchPool, 86
 - handle_serverPool, 86
- TYPE
 - request, 17
- types.h
 - LIST, 88
- uni_list.c
 - alloc_new_job, 100
 - dequeue, 100
 - dequeue_job, 100
 - enqueue, 100
 - enqueue_job, 100
 - exists_job_res, 100
 - init_job_queue, 100
 - new, 100
 - oldoutptfd, 101
 - push, 101
 - push_job, 101
 - remove_elem, 101
 - remove_job, 101
 - run_job, 101
 - update_list, 101
- uni_list.h
 - alloc_new_job, 90
 - container_of2, 90
 - dequeue, 90
 - dequeue_job, 90
 - enqueue, 90
 - enqueue_job, 90
 - exists_job_res, 90
 - init_job_queue, 90
 - init_list_head, 90
 - new_list, 91
 - push_job, 91
 - remove_elem, 91
 - remove_job, 91
 - run_job, 91
 - update_list, 91

UNKNOWN_ERROR
 proxy/include/consts.h, [33](#)

UNKNOWN_ERROR_SIZE
 proxy/include/consts.h, [33](#)

update_list
 uni_list.c, [101](#)
 uni_list.h, [91](#)

V
 lett_scritt.h, [48](#)

valore
 list_pprint, [14](#)

value
 hash_tbl, [9](#)

WHILE_GETOPT_L
 params.h, [78](#)

WHITE
 proxy/include/consts.h, [33](#)

whoami
 ARGS_ONE, [7](#)

WREAD
 lett_scritt.h, [48](#)

WRONG_REQUEST
 proxy/include/consts.h, [33](#)

WWRITE
 lett_scritt.h, [48](#)

YELLOW
 proxy/include/consts.h, [33](#)