Chatterbox Lorenzo Gazzella 546890

Generated by Doxygen 1.8.13

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 array_struct Struct Reference

Struttura utilizzata per memorizzare l'insieme degli utenti connessi.

`#include <struttureCondivise.h>`

Collaboration diagram for array_struct:

Data Fields

- utente_connesso_s * **arr**
- int **nConnessi**
- pthread_mutex_t **arr_m**

3.1.1 Detailed Description

Struttura utilizzata per memorizzare l'insieme degli utenti connessi.

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

- struttureCondivise.h

3.2 coda_circolare Struct Reference

Data Fields

- void ** **coda**
- int **testa**
- int **lung**
- int **maxlung**
- void(* **free**)(void *elem)

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

- lib/GestioneHistory/codaCircolare.h

3.3 coda_circolare_iteratore Struct Reference

Collaboration diagram for coda_circolare_iteratore:

Data Fields

- coda_circolare_s * **codaCircolare**
- int **next**
- int **daLeggere**

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

- lib/GestioneHistory/codaCircolare.h

3.4 config_s Struct Reference

Data Fields

- char * **UnixPath**
- int **MaxConnections**
- int **ThreadsInPool**
- int **MaxMsgSize**
- int **MaxFileSize**
- int **MaxHistMsgs**
- char * **DirName**
- char * **StatFileName**

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

- parser.h

3.5 connessi Struct Reference

Struttura utilizzata per memorizzare un contatore in mutua esclusione. Verrà utilizzato per tenere traccia del numero di socket attive.

```
#include <struttureCondivise.h>
```

Data Fields

- int **contatore**
- pthread_mutex_t **contatore_m**

3.5.1 Detailed Description

Struttura utilizzata per memorizzare un contatore in mutua esclusione. Verrà utilizzato per tenere traccia del numero di socket attive.

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

- struttureCondivise.h

3.6 data Struct Reference

body del messaggio

`#include <message.h>`

3.6.1 Detailed Description

body del messaggio

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

- message.h

3.7 hash Struct Reference

Struttura utilizzata per memorizzare tutti gli utenti registrati.

`#include <struttureCondivise.h>`

Collaboration diagram for hash:

Data Fields

- icl_hash_t * **hash**
- pthread_mutex_t **hash_m** [HASHSIZE/HASHGROUPSIZE]

3.7.1 Detailed Description

Struttura utilizzata per memorizzare tutti gli utenti registrati.

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

- struttureCondivise.h

3.8 header Struct Reference

header del messaggio

`#include <message.h>`

3.8.1 Detailed Description

header del messaggio

header della parte dati

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

- message.h

3.9 icl_entry_s Struct Reference

Collaboration diagram for icl_entry_s:

Data Fields

- void * **key**
- void * **data**
- struct icl_entry_s * **next**

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

- lib/GestioneHashTable/icl_hash.h

3.10 icl_hash_s Struct Reference

Collaboration diagram for icl_hash_s:

Data Fields

- int **nbuckets**
- int **nentries**
- icl_entry_t ** **buckets**
- unsigned int(* **hash_function**)(void *)
- int(* **hash_key_compare**)(void *, void *)

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

- lib/GestioneHashTable/icl_hash.h

3.11 message_data_hdr_t Struct Reference

Data Fields

- char **receiver** [MAX_NAME_LENGTH+1]
- unsigned int **len**

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

- message.h

3.12 message_data_t Struct Reference

Collaboration diagram for message_data_t:

Data Fields

- message_data_hdr_t **hdr**
- char * **buf**

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

- message.h

3.13 message_hdr_t Struct Reference

Data Fields

- op_t **op**
- char **sender** [MAX_NAME_LENGTH+1]

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

- message.h

3.14 message_t Struct Reference

Collaboration diagram for message_t:

Data Fields

- message_hdr_t **hdr**
- message_data_t **data**

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

- message.h

3.15 messaggio Struct Reference

tipo del messaggio

`#include <message.h>`

3.15.1 Detailed Description

tipo del messaggio

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

- message.h

3.16 Node Struct Reference

`#include <queue.h>`

Collaboration diagram for Node:

Data Fields

- void * **data**
- struct Node * **next**

3.16.1 Detailed Description

Elemento della coda.

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

- lib/GestioneQueue/queue.h

3.17 operation_t Struct Reference

Data Fields

- char * **sname**
- char * **rname**
- op_t **op**
- char * **msg**
- long **size**
- long **n**

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

- client.c

3.18 Queue Struct Reference

```
#include <queue.h>
```

Collaboration diagram for Queue:

Data Fields

- Node_t * **head**
- Node_t * **tail**
- unsigned long **qlen**

3.18.1 Detailed Description

Struttura dati coda.

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

- lib/GestioneQueue/queue.h

3.19 stat_struct Struct Reference

Struttura utilizzata per avere la struct delle statistiche in mutua esclusione.

```
#include <struttureCondivise.h>
```

Collaboration diagram for stat_struct:

Data Fields

- struct statistics **stat**
- pthread_mutex_t **stat_m**

3.19.1 Detailed Description

Struttura utilizzata per avere la struct delle statistiche in mutua esclusione.

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

- struttureCondivise.h

3.20 statistics Struct Reference

Data Fields

- unsigned long **nusers**
- unsigned long **nonline**
- unsigned long **ndelivered**
- unsigned long **nnotdelivered**
- unsigned long **nfiledelivered**
- unsigned long **nfilenotdelivered**
- unsigned long **nerrors**

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

- stats.h

3.21 utente_connesso Struct Reference

Struttura utilizzata per memorizzare tutte le informazioni di un utente connesso.

`#include <struttureCondivise.h>`

Data Fields

- char * **nickname**
- long **fd**
- pthread_mutex_t **fd_m**

3.21.1 Detailed Description

Struttura utilizzata per memorizzare tutte le informazioni di un utente connesso.

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

- struttureCondivise.h

Chapter 4

File Documentation

4.1 chatty.c File Reference

File principale del server chatterbox Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

```
#include "./chatty.h"
```

Include dependency graph for chatty.c:

4.2 chatty.h File Reference

File che contiene la dichiarazione di tutte le funzioni utilizzate nel file chatty.c Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <sys/un.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <getopt.h>
#include "./struttureCondivise.h"
#include "./parser.h"
#include "./lib/GestioneQueue/queue.h"
#include "./lib/GestioneHashTable/icl_hash.h"
#include "./stats.h"
#include "./utility.h"
#include "./connections.h"
#include "./config.h"
#include "./gestioneRichieste.h"
```

Include dependency graph for chatty.h: This graph shows which files directly or indirectly include this file:

Macros

- #define **_POSIX_C_SOURCE** 200809L

Functions

- int createSocket ()

  Funzione che si occupa della creazione della socket.
- int aggiorna (fd_set *set, int max)

  Funzione che trova file descriptor più grande all'intero di un insieme di fd. Verrà chiamata a seguito di una cancellazione di un fd dall'insieme degli fd.
- int isPipe (int fd)

  Funzione che verifica se un fd è una pipe.
- void **printRisOP** (message_t m, int ok)
- void stopAllThread (int segnali, int listener, int nThreadAttivi)

  Funzione che stoppa i thread indicati dai parametri.
- void joinAllThread ()

  Funzione effettua le join di tutti i thread del server.
- void stopPool (int nThreadAttivi)

  Funzione che stoppa un certo numero di thread del pool.
- int initHashLock ()

  Funzione che inizializza l'array (utentiRegistrati.hash_m) delle lock della tabella hash.
- int initRegistrati ()

  Funzione che inizializza l'intera hash table degli utenti registrati e le relative lock.
- int initDirFile ()

  Funzione che inizializza la directory in cui si andranno a salvare tutti i file scambiati tra gli utenti.
- int initNSock ()

  Funzione che inizializza il contatore di socket attive e la relativa lock. Se la cartella esiste già la svuota, altrimenti la crea.
- int initStat ()

  Funzione che inizializza la struct delle statistiche.
- int initConnessi ()

  Funzione che inizializza l'array degli utenti connessi e le relative lock.
- void cleanupConfigurazione ()

  Funzione che libera la memoria occupata dalla struct delle configurazioni.
- void cleanupRichieste ()

  Funzione che libera la memoria occupata dalla coda delle richieste.
- void cleanupStat ()

  Funzione che libera la memoria occupata dalla struct delle statistiche.
- void cleanupNSock ()

  Funzione che libera la memoria occupata dalla struct relativa al contatore di socket attive.
- void cleanupRegistrati ()

  Funzione che libera la memoria occupata hash degli utenti registrati.
- void cleanupConnessi ()

  Funzione che libera la memoria occupata dall'array degli utenti connessi.
- void cleanupPipe ()

  Funzione che libera la memoria occupata dall'array di pipe usate per la comunicazione tra listener e pool.
- void cleanupThreadId ()

  Funzione che libera la memoria occupata dall'array degli id dei thread.

4.2.1 Detailed Description

File che contiene la dichiarazione di tutte le funzioni utilizzate nel file chatty.c Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

Author

Lorenzo Gazzella 546890

4.2.2 Function Documentation

4.2.2.1 aggiorna()

```
int aggiorna (
            fd_set * set,
            int max )
```

Funzione che trova file descriptor più grande all'intero di un insieme di fd. Verrà chiamata a seguito di una cancellazione di un fd dall'insieme degli fd.

aggiorna

Parameters

set	Insieme di file descriptor
max	Massimo attuale.

Returns

Nuovo massimo

4.2.2.2 cleanupConfigurazione()

```
void cleanupConfigurazione ( )
```

Funzione che libera la memoria occupata dalla struct delle configurazioni.

cleanupConfigurazione

4.2.2.3 cleanupConnessi()

```
void cleanupConnessi ( )
```

Funzione che libera la memoria occupata dall'array degli utenti connessi.

cleanupConnessi

4.2.2.4 cleanupNSock()

```
void cleanupNSock ( )
```

Funzione che libera la memoria occupata dalla struct relativa al contatore di socket attive.

cleanupNSock

4.2.2.5 cleanupPipe()

```
void cleanupPipe ( )
```

Funzione che libera la memoria occupata dall'array di pipe usate per la comunicazione tra listener e pool.

cleanupPipe

4.2.2.6 cleanupRegistrati()

```
void cleanupRegistrati ( )
```

Funzione che libera la memoria occupata hash degli utenti registrati.

cleanupRegistrati

4.2.2.7 cleanupRichieste()

```
void cleanupRichieste ( )
```

Funzione che libera la memoria occupata dalla coda delle richieste.

cleanupRichieste

4.2.2.8 cleanupStat()

```
void cleanupStat ( )
```

Funzione che libera la memoria occupata dalla struct delle statistiche.

cleanupStat

4.2.2.9 cleanupThreadId()

```
void cleanupThreadId ( )
```

Funzione che libera la memoria occupata dall'array degli id dei thread.

cleanupThreadId

4.2.2.10 createSocket()

```
int createSocket ( )
```

Funzione che si occupa della creazione della socket.

createSocket

Returns

In caso di successo torna il file descriptor della nuova socket. In caso di errore torna -1

4.2.2.11 initConnessi()

```
int initConnessi ( )
```

Funzione che inizializza l'array degli utenti connessi e le relative lock.

initConnessi

Returns

0 in caso di successo, -1 altirmenti

4.2.2.12 initDirFile()

```
int initDirFile ( )
```

Funzione che inizializza la directory in cui si andranno a salvare tutti i file scambiati tra gli utenti.

initDirFile

Returns

0 in caso di successo, -1 altirmenti

4.2.2.13 initHashLock()

```
int initHashLock ( )
```

Funzione che inizializza l'array (utentiRegistrati.hash_m) delle lock della tabella hash.

initHashLock

Returns

0 in caso di successo, -1 altirmenti

4.2.2.14 initNSock()

```
int initNSock ( )
```

Funzione che inizializza il contatore di socket attive e la relativa lock. Se la cartella esiste già la svuota, altrimenti la crea.

initNSock

Returns

0 in caso di successo, -1 altirmenti

4.2.2.15 initRegistrati()

```
int initRegistrati ( )
```

Funzione che inizializza l'intera hash table degli utenti registrati e le relative lock.

initRegistrati

Returns

0 in caso di successo, -1 altirmenti

4.2.2.16 initStat()

```
int initStat ( )
```

Funzione che inizializza la struct delle statistiche.

initStat

Returns

0 in caso di successo, -1 altirmenti

4.2.2.17 isPipe()

```
int isPipe (
            int fd )
```

Funzione che verifica se un fd è una pipe.

isPipe

Parameters

fd	fd da verificare

Returns

: -1 se ci sono stati errori 1 se fd è una pipe 0 altrimenti

4.2.2.18 joinAllThread()

```
void joinAllThread ( )
```

Funzione effettua le join di tutti i thread del server.

joinAllThread

4.2.2.19 stopAllThread()

```
void stopAllThread (
            int segnali,
            int listener,
            int nThreadAttivi )
```

Funzione che stoppa i thread indicati dai parametri.

stopAllThread

Parameters

segnali	Varrà 1 se la funzione deve stoppare il thread dei segnali, 0 altirmenti
listener	Varrà 1 se la funzione deve stoppare il thread listener, 0 altirmenti
nThreadAttivi	Numero dei thread del pool che deve stoppare

4.2.2.20 stopPool()

```
void stopPool (
            int nThreadAttivi )
```

Funzione che stoppa un certo numero di thread del pool.

stopPool

Parameters

nThreadAttivi	Numero di thread che deve stoppare

4.3 client.c File Reference

Semplice client di test.

```
#include <time.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "connections.h"
#include "ops.h"
```

Include dependency graph for client.c:

Data Structures

- struct operation_t

Macros

- #define **_POSIX_C_SOURCE** 200809L

Functions

- int **main** (int argc, char ∗argv[ ])

4.3.1 Detailed Description

Semplice client di test.

4.4 config.h File Reference

File contenente alcune define con valori massimi utilizzabili.

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

Macros

- #define **MAX_NAME_LENGTH** 32
- #define **HASHSIZE** 1000
- #define **HASHGROUPSIZE** 250

Typedefs

- typedef int **make_iso_compilers_happy**

4.4.1 Detailed Description

File contenente alcune define con valori massimi utilizzabili.

4.5 gestioneRichieste.c File Reference

Contiene l'implementazione di tutti i metodi per la gestione delle singole operazioni che il server gestisce. Constiene inoltre alcune funzioni di utilità utilizzate nel corpo delle funzione per gestire le richieste. Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

```
#include "gestioneRichieste.h"
```

Include dependency graph for gestioneRichieste.c:

Functions

- char * makeListUsr ()

  Funzione che restiruisce la lista degli utenti connessi.
- int registrato (char *nickname)

  Funzione boolena che controlla s eun determinato nickname è presente nella hash table dei registrati.
- message_t * copyMessage (message_t m)

  Funzione che esegue la copia di un messaggio.
- int getUsrNickname (char *nickname)

  Funzione che restituisce la posizione di un nickname all'interno dell'array dei connessi.
- int getUsrFd (long fd)

  Funzione che restituisce la posizione di un fd all'interno dell'array dei connessi.
- char * getOnlyFileName (char *path)

  Funzione che dato un path, restituisce il nome del file associato al path.
- int getPosizioneLibera ()

  Funzione che restituisce prima posizione libera all'interno dell'array dei connessi.
- int register_op (long fd, message_t m)

  Funzione che gestisce la richiesta di registrazione di un client. Il client verrà automaticamente connesso e riceverà la lista degli utenti online.
- int connect_op (long fd, message_t m, int atomica)

  Funzione che gestisce la richiesta di connessione di un client. Il client riceverà automaticamente la lista degli utenti online.
- int usrlist_op (long fd, message_t m, int atomica)

  Funzione che invia la lista degli utenti connessi al client che ne fa richiesta.
- int unregister_op (long fd, message_t m)

  Funzione che gestisce la richiesta di deregistrazione da parte di un client.
- int disconnect_op (long fd)

  Funzione che gestisce la richiesta di disconnessione da parte di un client.
- int posttxt_op (long fd, message_t m)

  Funzione che gestisce la richiesta di invio di un messaggio da parte di un client a un altro. Il messaggio sarà inviato direttamente al client destinatario nel caso in cui sia connesso. In ogni caso il messaggio verrà aggiunto alla history del client destinatario.

- int getprevmsgs_op (long fd, message_t m)

  Funzione che gestisce l'invio della history relativa al client che ne ha fatto richiesta.
- int posttxtall_op (long fd, message_t m)

  Funzione che gestisce la richiesta da parte di un client dell'invio a tutti gli utenti registrati di un certo messaggio. L'invio del messaggio segue le stesse regole della funzione posttxt_op.
- int postfile_op (long fd, message_t m)

  Funzione che gestisce la richiesta di invio di un file da parte di un client a un altro. Il client destinatario sarà notificato nel caso in cui sia connesso. In ogni caso verrà aggiunto alla history del client destinatario il nome del file.
- int getfile_op (long fd, message_t m)

  Funzione che gestisce la richiesta da parte di un client del download di un file.

4.5.1 Detailed Description

Contiene l'implementazione di tutti i metodi per la gestione delle singole operazioni che il server gestisce. Constiene inoltre alcune funzioni di utilità utilizzate nel corpo delle funzione per gestire le richieste. Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

Author

Lorenzo Gazzella 546890

4.5.2 Function Documentation

4.5.2.1 connect_op()

```
int connect_op (
            long fd,
            message_t m,
            int atomica )
```

Funzione che gestisce la richiesta di connessione di un client. Il client riceverà automaticamente la lista degli utenti online.

connect_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client
atomica	Variabile booleana. Deve valere 1 se si invoca con la lock sulla hash degli utenti registrati, 0 altirmenti.

Returns

0 in caso di successo -1 in caso di errore

4.5.2.2 copyMessage()

```
message_t * copyMessage (
            message_t m )
```

Funzione che esegue la copia di un messaggio.

copyMessage

Parameters

m	Messaggio da copiare

Returns

Nuovo messaggio in caso di successo. NULL altrimenti

4.5.2.3 disconnect_op()

```
int disconnect_op (
            long fd )
```

Funzione che gestisce la richiesta di disconnessione da parte di un client.

register_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.5.2.4 getfile_op()

```
int getfile_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta da parte di un client del download di un file.

getfile_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.5.2.5 getOnlyFileName()

```
char * getOnlyFileName (
            char * path )
```

Funzione che dato un path, restituisce il nome del file associato al path.

getOnlyFileName

Parameters

path	Path da controllare

Returns

Nome del file associato al path

4.5.2.6 getPosizioneLibera()

```
int getPosizioneLibera ( )
```

Funzione che restituisce prima posizione libera all'interno dell'array dei connessi.

getPosizioneLibera

Parameters

fd	fd da cercare

Returns

La posizone in caso di successo. -1 altrimenti

4.5.2.7 getprevmsgs_op()

```
int getprevmsgs_op (
            long fd,
            message_t m )
```

Funzione che gestisce l'invio della history relativa al client che ne ha fatto richiesta.

getprevmsgs_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.5.2.8 getUsrFd()

```
int getUsrFd (
            long fd )
```

Funzione che restituisce la posizione di un fd all'interno dell'array dei connessi.

getUsrFd

Parameters

fd	fd da cercare

Returns

Se esiste un utente connesso con quel fd, restituisce la sua posizone. -1 altrimenti

4.5.2.9 getUsrNickname()

```
int getUsrNickname (
            char * nickname )
```

Funzione che restituisce la posizione di un nickname all'interno dell'array dei connessi.

getUsrNickname

Parameters

nickname	Nickname da cercare

Returns

Se esiste un utente connesso con quel nickname, restituisce la sua posizone. -1 altrimenti

4.5.2.10 makeListUsr()

```
char * makeListUsr ( )
```

Funzione che restiruisce la lista degli utenti connessi.

makeListUsr

Returns

Lista utenti connessi

4.5.2.11 postfile_op()

```
int postfile_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta di invio di un file da parte di un client a un altro. Il client destinatario sarà notificato nel caso in cui sia connesso. In ogni caso verrà aggiunto alla history del client destinatario il nome del file.

postfile_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.5.2.12 posttxt_op()

```
int posttxt_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta di invio di un messaggio da parte di un client a un altro. Il messaggio sarà inviato direttamente al client destinatario nel caso in cui sia connesso. In ogni caso il messaggio verrà aggiunto alla history del client destinatario.

posttxt_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.5.2.13 posttxtall_op()

```
int posttxtall_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta da parte di un client dell'invio a tutti gli utenti registrati di un certo messaggio. L'invio del messaggio segue le stesse regole della funzione posttxt_op.

posttxtall_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.5.2.14 register_op()

```
int register_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta di registrazione di un client. Il client verrà automaticamente connesso e riceverà la lista degli utenti online.

register_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.5.2.15 registrato()

```
int registrato (
            char * nickname )
```

Funzione boolena che controlla s eun determinato nickname è presente nella hash table dei registrati.

registrato

Parameters

nickname	Nickname da controllare

Returns

1 nel caso in cui sia registrato. 0 altrimenti

4.5.2.16 unregister_op()

```
int unregister_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta di deregistrazione da parte di un client.

register_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.5.2.17 usrlist_op()

```
int usrlist_op (
            long fd,
            message_t m,
            int atomica )
```

Funzione che invia la lista degli utenti connessi al client che ne fa richiesta.

usrlist_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client
atomica	Variabile booleana. Deve valere 1 se si invoca con la lock sulla hash degli utenti registrati, con la lock sull'array degli utenti connessi e con la lock sull'fd, 0 altirmenti.

Returns

0 in caso di successo -1 in caso di errore

4.6 gestioneRichieste.h File Reference

Contiene la dichiarazione di tutti i metodi per la gestione delle singole operazioni che il server gestisce. Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <strings.h>
#include "message.h"
#include "ops.h"
#include "./config.h"
#include "./struttureCondivise.h"
#include "./gestioneRichieste.h"
#include "./chatty.h"
#include "./parser.h"
#include "./connections.h"
#include "./lib/GestioneHistory/codaCircolare.h"
```

Include dependency graph for gestioneRichieste.h: This graph shows which files directly or indirectly include this file:

Macros

- #define **_POSIX_C_SOURCE** 200809L
- #define **_GNU_SOURCE**

Functions

- int register_op (long fd, message_t m)

  Funzione che gestisce la richiesta di registrazione di un client. Il client verrà automaticamente connesso e riceverà la lista degli utenti online.
- int connect_op (long fd, message_t m, int atomica)

  Funzione che gestisce la richiesta di connessione di un client. Il client riceverà automaticamente la lista degli utenti online.
- int posttxt_op (long fd, message_t m)

  Funzione che gestisce la richiesta di invio di un messaggio da parte di un client a un altro. Il messaggio sarà inviato direttamente al client destinatario nel caso in cui sia connesso. In ogni caso il messaggio verrà aggiunto alla history del client destinatario.
- int posttxtall_op (long fd, message_t m)

  Funzione che gestisce la richiesta da parte di un client dell'invio a tutti gli utenti registrati di un certo messaggio. L'invio del messaggio segue le stesse regole della funzione posttxt_op.
- int postfile_op (long fd, message_t m)

  Funzione che gestisce la richiesta di invio di un file da parte di un client a un altro. Il client destinatario sarà notificato nel caso in cui sia connesso. In ogni caso verrà aggiunto alla history del client destinatario il nome del file.
- int getfile_op (long fd, message_t m)

  Funzione che gestisce la richiesta da parte di un client del download di un file.
- int getprevmsgs_op (long fd, message_t m)

  Funzione che gestisce l'invio della history relativa al client che ne ha fatto richiesta.
- int usrlist_op (long fd, message_t m, int atomica)

  Funzione che invia la lista degli utenti connessi al client che ne fa richiesta.
- int unregister_op (long fd, message_t m)

  Funzione che gestisce la richiesta di deregistrazione da parte di un client.
- int disconnect_op (long fd)

  Funzione che gestisce la richiesta di disconnessione da parte di un client.

4.6.1 Detailed Description

Contiene la dichiarazione di tutti i metodi per la gestione delle singole operazioni che il server gestisce. Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

Author

Lorenzo Gazzella 546890

4.6.2 Function Documentation

4.6.2.1 connect_op()

```
int connect_op (
            long fd,
            message_t m,
            int atomica )
```

Funzione che gestisce la richiesta di connessione di un client. Il client riceverà automaticamente la lista degli utenti online.

connect_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client
atomica	Variabile booleana. Deve valere 1 se si invoca con la lock sulla hash degli utenti registrati, 0 altirmenti.

Returns

0 in caso di successo -1 in caso di errore

4.6.2.2 disconnect_op()

```
int disconnect_op (
            long fd )
```

Funzione che gestisce la richiesta di disconnessione da parte di un client.

register_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.6.2.3 getfile_op()

```
int getfile_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta da parte di un client del download di un file.

getfile_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.6.2.4 getprevmsgs_op()

```
int getprevmsgs_op (
            long fd,
            message_t m )
```

Funzione che gestisce l'invio della history relativa al client che ne ha fatto richiesta.

getprevmsgs_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.6.2.5 postfile_op()

```
int postfile_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta di invio di un file da parte di un client a un altro. Il client destinatario sarà notificato nel caso in cui sia connesso. In ogni caso verrà aggiunto alla history del client destinatario il nome del file.

postfile_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.6.2.6 posttxt_op()

```
int posttxt_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta di invio di un messaggio da parte di un client a un altro. Il messaggio sarà inviato direttamente al client destinatario nel caso in cui sia connesso. In ogni caso il messaggio verrà aggiunto alla history del client destinatario.

posttxt_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.6.2.7 posttxtall_op()

```
int posttxtall_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta da parte di un client dell'invio a tutti gli utenti registrati di un certo messaggio. L'invio del messaggio segue le stesse regole della funzione posttxt_op.

posttxtall_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.6.2.8 register_op()

```
int register_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta di registrazione di un client. Il client verrà automaticamente connesso e riceverà la lista degli utenti online.

register_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.6.2.9 unregister_op()

```
int unregister_op (
            long fd,
            message_t m )
```

Funzione che gestisce la richiesta di deregistrazione da parte di un client.

register_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client

Returns

0 in caso di successo -1 in caso di errore

4.6.2.10 usrlist_op()

```
int usrlist_op (
            long fd,
            message_t m,
            int atomica )
```

Funzione che invia la lista degli utenti connessi al client che ne fa richiesta.

usrlist_op

Parameters

fd	Fd del client che sta facendo la richiesta
m	Messaggio inviato dal client
atomica	Variabile booleana. Deve valere 1 se si invoca con la lock sulla hash degli utenti registrati, con la lock sull'array degli utenti connessi e con la lock sull'fd, 0 altirmenti.

Returns

0 in caso di successo -1 in caso di errore

4.7 lib/GestioneHashTable/icl_hash.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include "icl_hash.h"
#include <limits.h>
```

Include dependency graph for icl_hash.c:

Macros

- #define **BITS_IN_int** (sizeof(int) ∗ CHAR_BIT)
- #define **THREE_QUARTERS** ((int) ((BITS_IN_int ∗ 3) / 4))
- #define **ONE_EIGHTH** ((int) (BITS_IN_int / 8))
- #define **HIGH_BITS** (∼((unsigned int)(∼0) >> ONE_EIGHTH))

Functions

- icl_hash_t ∗ icl_hash_create (int nbuckets, unsigned int(∗hash_function)(void ∗), int(∗hash_key_↩ compare)(void ∗, void ∗))
- void ∗ icl_hash_find (icl_hash_t ∗ht, void ∗key)
- icl_entry_t ∗ icl_hash_insert (icl_hash_t ∗ht, void ∗key, void ∗data)
- int icl_hash_delete (icl_hash_t ∗ht, void ∗key, void(∗free_key)(void ∗), void(∗free_data)(void ∗))
- int icl_hash_destroy (icl_hash_t ∗ht, void(∗free_key)(void ∗), void(∗free_data)(void ∗))
- int icl_hash_dump (FILE ∗stream, icl_hash_t ∗ht)

4.7.1 Detailed Description

Dependency free hash table implementation.

This simple hash table implementation should be easy to drop into any other peice of code, it does not depend on anything else :-)

Author

Jakub Kurzak

4.7.2 Function Documentation

4.7.2.1 icl_hash_create()

```
icl_hash_t* icl_hash_create (
            int nbuckets,
            unsigned int(*)(void *) hash_function,
            int(*)(void *, void *) hash_key_compare )
```

Create a new hash table.

Parameters

in	*nbuckets*	– number of buckets to create
in	*hash_function*	– pointer to the hashing function to be used
in	*hash_key_compare*	– pointer to the hash key comparison function to be used

Returns

pointer to new hash table.

4.7.2.2 icl_hash_delete()

```
int icl_hash_delete (
            icl_hash_t * ht,
            void * key,
            void(*)(void *) free_key,
            void(*)(void *) free_data )
```

Free one hash table entry located by key (key and data are freed using functions).

Parameters

ht	– the hash table to be freed
key	– the key of the new item
free_key	– pointer to function that frees the key
free_data	– pointer to function that frees the data

Returns

0 on success, -1 on failure.

4.7.2.3 icl_hash_destroy()

```
int icl_hash_destroy (
            icl_hash_t * ht,
            void(*)(void *) free_key,
            void(*)(void *) free_data )
```

Free hash table structures (key and data are freed using functions).

Parameters

ht	– the hash table to be freed
free_key	– pointer to function that frees the key
free_data	– pointer to function that frees the data

Returns

0 on success, -1 on failure.

4.7.2.4 icl_hash_dump()

```
int icl_hash_dump (
            FILE * stream,
            icl_hash_t * ht )
```

Dump the hash table's contents to the given file pointer.

Parameters

stream	– the file to which the hash table should be dumped
ht	– the hash table to be dumped

Returns

0 on success, -1 on failure.

4.7.2.5 icl_hash_find()

```
void* icl_hash_find (
            icl_hash_t * ht,
            void * key )
```

Search for an entry in a hash table.

Parameters

ht	– the hash table to be searched
key	– the key of the item to search for

Returns

pointer to the data corresponding to the key. If the key was not found, returns NULL.

4.7.2.6 icl_hash_insert()

```
icl_entry_t* icl_hash_insert (
             icl_hash_t * ht,
             void * key,
             void * data )
```

Insert an item into the hash table.

Parameters

ht	– the hash table
key	– the key of the new item
data	– pointer to the new item's data

Returns

pointer to the new item. Returns NULL on error.

4.8 lib/GestioneHashTable/icl_hash.h File Reference

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

Include dependency graph for icl_hash.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct icl_entry_s
- struct icl_hash_s

Macros

- #define **icl_hash_foreach**(ht, tmpint, tmpent, kp, dp)

Typedefs

- typedef struct icl_entry_s **icl_entry_t**
- typedef struct icl_hash_s **icl_hash_t**

Functions

- icl_hash_t * icl_hash_create (int nbuckets, unsigned int(*hash_function)(void *), int(*hash_key_↩ compare)(void *, void *))
- void * icl_hash_find (icl_hash_t *, void *)
- icl_entry_t * icl_hash_insert (icl_hash_t *, void *, void *)
- int **icl_hash_destroy** (icl_hash_t *, void(*)(void *), void(*)(void *))
- int icl_hash_dump (FILE *, icl_hash_t *)
- int icl_hash_delete (icl_hash_t *ht, void *key, void(*free_key)(void *), void(*free_data)(void *))

4.8.1 Detailed Description

Header file for icl_hash routines.

4.8.2 Macro Definition Documentation

4.8.2.1 icl_hash_foreach

```
#define icl_hash_foreach(
              ht,
              tmpint,
              tmpent,
              kp,
              dp )
```

Value:

```
for (tmpint=0;tmpint<ht->nbuckets; tmpint++)        \
       for (tmpent=ht->buckets[tmpint];                                  \
            tmpent!=NULL&&((kp=tmpent->key)!=NULL)&&((dp=tmpent->data)!=NULL); \
            tmpent=tmpent->next)
```

4.8.3 Function Documentation

4.8.3.1 icl_hash_create()

```
icl_hash_t* icl_hash_create (
             int nbuckets,
             unsigned int(*)(void *) hash_function,
             int(*)(void *, void *) hash_key_compare )
```

Create a new hash table.

Parameters

in	*nbuckets*	– number of buckets to create
in	*hash_function*	– pointer to the hashing function to be used
in	*hash_key_compare*	– pointer to the hash key comparison function to be used

Returns

pointer to new hash table.

4.8.3.2 icl_hash_delete()

```
int icl_hash_delete (
            icl_hash_t * ht,
            void * key,
            void(*)(void *) free_key,
            void(*)(void *) free_data )
```

Free one hash table entry located by key (key and data are freed using functions).

Parameters

ht	– the hash table to be freed
key	– the key of the new item
free_key	– pointer to function that frees the key
free_data	– pointer to function that frees the data

Returns

0 on success, -1 on failure.

4.8.3.3 icl_hash_dump()

```
int icl_hash_dump (
            FILE * stream,
            icl_hash_t * ht )
```

Dump the hash table's contents to the given file pointer.

Parameters

stream	– the file to which the hash table should be dumped
ht	– the hash table to be dumped

Returns

0 on success, -1 on failure.

4.8.3.4 icl_hash_find()

```
void* icl_hash_find (
            icl_hash_t * ht,
            void * key )
```

Search for an entry in a hash table.

Parameters

ht	– the hash table to be searched
key	– the key of the item to search for

Returns

pointer to the data corresponding to the key. If the key was not found, returns NULL.

4.8.3.5 icl_hash_insert()

```
icl_entry_t* icl_hash_insert (
            icl_hash_t * ht,
            void * key,
            void * data )
```

Insert an item into the hash table.

Parameters

ht	– the hash table
key	– the key of the new item
data	– pointer to the new item's data

Returns

pointer to the new item. Returns NULL on error.

4.9 lib/GestioneHistory/codaCircolare.h File Reference

File che contiene la dichiarazione delle funzioni per la gestione di una coda circolare Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

`#include <stdlib.h>`
Include dependency graph for codaCircolare.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct coda_circolare
- struct coda_circolare_iteratore

Typedefs

- typedef struct coda_circolare **coda_circolare_s**
- typedef struct coda_circolare_iteratore **coda_circolare_iteratore_s**

Functions

- coda_circolare_s * initCodaCircolare (int maxlung, void(*f)(void *elem))
  Funzione che inizializza una coda circolare di una determinata lunghezza.
- int vuota (coda_circolare_s *c)
  Funzione booleana utilizzata per controllare se la coda è vuota.
- int lung (coda_circolare_s *c)
  Funzione restiruisce il numero di elementi nella coda.
- int inserisci (coda_circolare_s *c, void *elem)
  Funzione che inserisce un elemento in fondo a una coda circolare.
- int elimina (coda_circolare_s *c)
  Funzione che elimina un elemento in testa ad una coda circolare.
- int eliminaCoda (coda_circolare_s *c)
  Funzione elimina una coda circolare.
- coda_circolare_iteratore_s * initIteratore (coda_circolare_s *c)
  Funzione inizializza un iteratore sulla coda circolare.
- void * next (coda_circolare_iteratore_s *i)
  Funzione che restituisce un elemento della coda circolare.
- void eliminaIteratore (coda_circolare_iteratore_s *i)
  Funzione che elimina un iteratore.

4.9.1 Detailed Description

File che contiene la dichiarazione delle funzioni per la gestione di una coda circolare Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

Author

Lorenzo Gazzella 546890

4.9.2 Function Documentation

4.9.2.1 elimina()

```
int elimina (
            coda_circolare_s * c )
```

Funzione che elimina un elemento in testa ad una coda circolare.

elimina

Parameters

c	Coda circolare da modificare

Returns

1 in caso di successo, 0 altrimenti

4.9.2.2 eliminaCoda()

```
int eliminaCoda (
            coda_circolare_s * c )
```

Funzione elimina una coda circolare.

eliminaCoda

Parameters

c	Coda circolare da eliminare

Returns

1 in caso di successo, 0 altrimenti

4.9.2.3 eliminaIteratore()

```
void eliminaIteratore (
            coda_circolare_iteratore_s * i )
```

Funzione che elimina un iteratore.

eliminaIteratore

Parameters

i	Iteratore

Returns

Iteratore da eliminare

4.9.2.4 initCodaCircolare()

```
coda_circolare_s* initCodaCircolare (
            int maxlung,
            void(*)(void *elem) f )
```

Funzione che inizializza una coda circolare di una determinata lunghezza.

initCodaCircolare

Parameters

maxlung	Lunghezza coda circolare
f	Funzione che libera la memoria occupata da un elemento all'interno della coda circolare

Returns

Coda circolare inizializzata

4.9.2.5 initIteratore()

```
coda_circolare_iteratore_s* initIteratore (
            coda_circolare_s * c )
```

Funzione inizializza un iteratore sulla coda circolare.

initIteratore

Parameters

c	Coda circolare da modificare

Returns

Iteratore sulla coda circolare inizializzato

4.9.2.6 inserisci()

```
int inserisci (
            coda_circolare_s * c,
            void * elem )
```

Funzione che inserisce un elemento in fondo a una coda circolare.

inserisci

Parameters

c	Coda circolare da modificare
elem	Elemento da inserire

Returns

1 in caso di successo, 0 altrimenti

4.9.2.7 lung()

```
int lung (
            coda_circolare_s * c )
```

Funzione restiruisce il numero di elementi nella coda.

lung

Parameters

c	Coda circolare da controllare

Returns

Numero di elementi nella coda

4.9.2.8 next()

```
void* next (
            coda_circolare_iteratore_s * i )
```

Funzione che restituisce un elemento della coda circolare.

next

Parameters

i	Iteratore

Returns

Elemento della coda circolare

4.9.2.9 vuota()

```
int vuota (
            coda_circolare_s * c )
```

Funzione booleana utilizzata per controllare se la coda è vuota.

vuota

Parameters

c	Coda circolare da controllare

Returns

1 nel caso in cui la coda sia vuota, 0 altirmenti

4.10 message.h File Reference

Contiene il formato del messaggio.

```
#include <assert.h>
#include <string.h>
#include "config.h"
#include "ops.h"
```

Include dependency graph for message.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct message_hdr_t
- struct message_data_hdr_t
- struct message_data_t
- struct message_t

4.10.1 Detailed Description

Contiene il formato del messaggio.

4.11 ops.h File Reference

Contiene i codici delle operazioni di richiesta e risposta.

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

Enumerations

- enum op_t {
  REGISTER_OP = 0, CONNECT_OP = 1, POSTTXT_OP = 2, POSTTXTALL_OP = 3,
  POSTFILE_OP = 4, GETFILE_OP = 5, GETPREVMSGS_OP = 6, USRLIST_OP = 7,
  UNREGISTER_OP = 8, DISCONNECT_OP = 9, OP_OK = 20, **TXT_MESSAGE** = 21,
  FILE_MESSAGE = 22, **OP_FAIL** = 25, **OP_NICK_ALREADY** = 26, **OP_NICK_UNKNOWN** = 27,
  OP_MSG_TOOLONG = 28, **OP_NO_SUCH_FILE** = 29, **OP_END** = 100 }

4.11.1 Detailed Description

Contiene i codici delle operazioni di richiesta e risposta.

4.11.2 Enumeration Type Documentation

4.11.2.1 op_t

`enum op_t`

Enumerator

CONNECT_OP	richiesta di registrazione di un ninckname
POSTTXT_OP	richiesta di connessione di un client
POSTTXTALL_OP	richiesta di invio di un messaggio testuale ad un nickname o groupname
POSTFILE_OP	richiesta di invio di un messaggio testuale a tutti gli utenti
GETFILE_OP	richiesta di invio di un file ad un nickname o groupname
GETPREVMSGS_OP	richiesta di recupero di un file
USRLIST_OP	richiesta di recupero della history dei messaggi
UNREGISTER_OP	richiesta di avere la lista di tutti gli utenti attualmente connessi
DISCONNECT_OP	richiesta di deregistrazione di un nickname o groupname
OP_OK	richiesta di disconnessione

4.12 parser.h File Reference

File che contiene tutte le funzioni necessarie al parsing di un file di configurazione Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

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

Data Structures

- struct config_s

Typedefs

- typedef struct config_s **config**

Functions

- int parse (char *nomeFile, config *configs)

  Prende un file e inserisce in configs tutte le configurazioni presenti nel file. Se ci sono due parametri uguali nello stesso file di configurazione, verrà preso il valore dell'ultimo.
- int makeConfig (char *parametro, char *value, config *configs)

  Funzione setta a un certo campo della struct un determinato valore.
- void RemoveSpaces (char *source)

  *Funzione che elimina gli tutti gli spazi, gli " e gli '
  ' da una determinata stringa.*
- int initParseCheck (char *pathFile, config *configs)

  Funzione che chiama in ordine: init, parse, check.
- int check (config *configs)

  Funzione che controlla se i valori contenuti in una struct sono validi.
- void init (config *configs)

  Funzione che inizializza i valori di una determinata structs.
- void FreeConfig (config *configs)

  Funzionen che libera un'intera struct config.

4.12.1 Detailed Description

File che contiene tutte le funzioni necessarie al parsing di un file di configurazione Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

Author

Lorenzo Gazzella 546890

4.12.2 Function Documentation

4.12.2.1 check()

```
int check (
            config * configs )
```

Funzione che controlla se i valori contenuti in una struct sono validi.

check

Parameters

configs	Struct che si vuole controllare

Returns

0 se non ci sono stati errori -1 se ci sono stati errori

4.12.2.2 FreeConfig()

```
void FreeConfig (
            config * configs )
```

Funzionen che libera un'intera struct config.

FreeConfig

Parameters

configs	Struct che si vuole liberare

4.12.2.3 init()

```
void init (
            config * configs )
```

Funzione che inizializza i valori di una determinata structs.

init

Parameters

configs	Struct che si vuole inizializzare

4.12.2.4 initParseCheck()

```
int initParseCheck (
            char * pathFile,
            config * configs )
```

Funzione che chiama in ordine: init, parse, check.

initParseCheck

Parameters

pathFile	Path del file che si vuole parsare
configs	Struct che si vuole modificare

Returns

0 se non ci sono stati errori -1 se ci sono stati errori

4.12.2.5 makeConfig()

```
int makeConfig (
            char * parametro,
            char * value,
            config * configs )
```

Funzione setta a un certo campo della struct un determinato valore.

makeConfig

Parameters

parametro	Campo della struct che si vuole modificare
value	Valore che si vuole settare al campo
configs	Struct che si vuole modificare

Returns

0 se non ci sono stati errori -1 se ci sono stati errori

4.12.2.6 parse()

```
int parse (
            char * nomeFile,
            config * configs )
```

Prende un file e inserisce in configs tutte le configurazioni presenti nel file. Se ci sono due parametri uguali nello stesso file di configurazione, verrà preso il valore dell'ultimo.

parse

Parameters

nomeFile	Path del file che si vuole parsare
configs	Struttura dati su cui si vogliono memorizzare i valori parsati

Returns

0 se non ci sono stati errori -1 se ci sono stati errori

4.12.2.7 RemoveSpaces()

```
void RemoveSpaces (
            char * source )
```

Funzione che elimina gli tutti gli spazi, gli " e gli '
' da una determinata stringa.

check

Parameters

configs	Struct che si vuole controllare

Returns

0 se non ci sono stati errori -1 se ci sono stati errori

4.13 struttureCondivise.h File Reference

File che contiene tutte le trutture dati utilizzate dal server Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

```
#include <pthread.h>
#include <string.h>
#include "./lib/GestioneQueue/queue.h"
#include "./lib/GestioneHashTable/icl_hash.h"
#include "./lib/GestioneHistory/codaCircolare.h"
#include "./message.h"
#include "./config.h"
#include "./parser.h"
#include "./stats.h"
```

Include dependency graph for struttureCondivise.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct utente_connesso

  Struttura utilizzata per memorizzare tutte le informazioni di un utente connesso.
- struct array_struct

  Struttura utilizzata per memorizzare l'insieme degli utenti connessi.
- struct connessi

  Struttura utilizzata per memorizzare un contatore in mutua esclusione. Verrà utilizzato per tenere traccia del numero di socket attive.
- struct hash

  Struttura utilizzata per memorizzare tutti gli utenti registrati.
- struct stat_struct

  Struttura utilizzata per avere la struct delle statistiche in mutua esclusione.

Typedefs

- typedef struct utente_connesso **utente_connesso_s**
- typedef struct array_struct **array_s**
- typedef struct connessi **connessi_s**
- typedef struct hash **hash_s**
- typedef struct stat_struct **stat_s**

Variables

- Queue_t * **richieste**
- config * **configurazione**
- hash_s * **utentiRegistrati**
- array_s * **utentiConnessi**
- connessi_s * **nSock**
- stat_s * **statistiche**

4.13.1 Detailed Description

File che contiene tutte le trutture dati utilizzate dal server Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

Author

Lorenzo Gazzella 546890

4.14 utility.h File Reference

Contiene alcune funzioni di utilità. Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

```
#include <errno.h>
#include <unistd.h>
```

Include dependency graph for utility.h: This graph shows which files directly or indirectly include this file:

Macros

- #define ec_meno1(s, m)

  Macro che controlla se un certo valore s è uguale a -1. Se è uguale stampa un messaggio di errore e termina.
- #define ec_null(s, m)

  Macro che controlla se un certo valore s è uguale a NULL. Se è uguale stampa un messaggio di errore e termina.
- #define ec_null_return(s, m)

  Macro che controlla se un certo valore s è uguale a NULL. Se è uguale stampa un messaggio di errore e ritorna -1.
- #define ec_meno1_return(s, m)

  Macro che controlla se un certo valore s è uguale a -1. Se è uguale stampa un messaggio di errore e ritorna -1.
- #define ec_meno1_c(s, m, c)

  Macro che controlla se un certo valore s è uguale a -1. Se è uguale stampa un messaggio di errore, esegue un comando e termina.
- #define ec_null_c(s, m, c)

  Macro che controlla se un certo valore s è uguale a NULL. Se è uguale stampa un messaggio di errore, esegue un comando e termina.

4.14.1 Detailed Description

Contiene alcune funzioni di utilità. Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore.

Author

Lorenzo Gazzella 546890

4.14.2 Macro Definition Documentation

4.14.2.1 ec_meno1

```
#define ec_meno1(
                s,
                m )
```

Value:

```
if ((s) == - 1) {\
    perror(m);\
    exit(EXIT_FAILURE);\
}
```

Macro che controlla se un certo valore s è uguale a -1. Se è uguale stampa un messaggio di errore e termina.

4.14.2.2 ec_meno1_c

```
#define ec_meno1_c(
                s,
                m,
                c )
```

Value:

```
if ((s) == -1) {\
    perror(m);\
    c;\
}
```

Macro che controlla se un certo valore s è uguale a -1. Se è uguale stampa un messaggio di errore, esegue un comando e termina.

4.14.2.3 ec_meno1_return

```
#define ec_meno1_return(
                s,
                m )
```

Value:

```
if ((s) == -1) {\
    perror(m);\
    return -1;\
}
```

Macro che controlla se un certo valore s è uguale a -1. Se è uguale stampa un messaggio di errore e ritorna -1.

4.14.2.4 ec_null

```
#define ec_null(
                s,
                m )
```

Value:

```
if ((s) == NULL) {\
    perror(m);\
    exit(EXIT_FAILURE);\
}
```

Macro che controlla se un certo valore s è uguale a NULL. Se è uguale stampa un messaggio di errore e termina.

4.14.2.5 ec_null_c

```
#define ec_null_c(
                s,
                m,
                c )
```

Value:

```
if ((s) == NULL) {\
    perror(m);\
    c;\
}
```

Macro che controlla se un certo valore s è uguale a NULL. Se è uguale stampa un messaggio di errore, esegue un comando e termina.

4.14.2.6 ec_null_return

```
#define ec_null_return(
                s,
                m )
```

Value:

```
if ((s) == NULL) {\
    perror(m);\
    return -1;\
}
```

Macro che controlla se un certo valore s è uguale a NULL. Se è uguale stampa un messaggio di errore e ritorna -1.

Index