chatty

0.1

Generato da Doxygen 1.8.14

Indice

1 Indice delle strutture dati

1.1 Strutture dati

Queste sono le strutture dati con una loro breve descrizione:

2 Indice dei file

2.1 Elenco dei file

Questo è un elenco dei file documentati con una loro breve descrizione:

3 Documentazione delle classi

3.1 Riferimenti per la struct conf_var

struttura per memorizzare variabili di configurazione

`#include <parser.h>`

Campi

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

3.1.1 Descrizione dettagliata

struttura per memorizzare variabili di configurazione

3.1.2 Documentazione dei campi

3.1.2.1 DirName

`DirName`

cartella dove salvare file di statistiche

3.1.2.2 MaxConnections

`MaxConnections`

numero massimo di connessioni simultanee

3.1.2.3 MaxFileSize

`MaxFileSize`

size massima per i file

3.1.2.4 MaxHistMsgs

`MaxHistMsgs`

size massima per history messaggi

3.1.2.5 MaxMsgSize

`MaxMsgSize`

dimensione massima dei messaggi

3.1.2.6 StatFileName

`StatFileName`

nome file di statistiche

3.1.2.7 ThreadsInPool

`ThreadsInPool`

numero di thread del server

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

- parser.h

3.2 Riferimenti per la struct icl_entry_t

Campi

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

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

- icl_hash.h

3.3 Riferimenti per la struct icl_hash_t

Campi

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

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

- icl_hash.h

3.4 Riferimenti per la struct message_data_hdr_t

header della parte dati

```
#include <message.h>
```

Campi

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

3.4.1 Descrizione dettagliata

header della parte dati

3.4.2 Documentazione dei campi

3.4.2.1 len

`len`

lunghezza del buffer dati

3.4.2.2 receiver

`receiver`

nickname del ricevente

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

- message.h

3.5 Riferimenti per la struct message_data_t

body del messaggio

`#include <message.h>`

Campi

- message_data_hdr_t hdr
- char * buf

3.5.1 Descrizione dettagliata

body del messaggio

3.5.2 Documentazione dei campi

3.5.2.1 buf

`buf`

buffer dati

3.5.2.2 hdr

`hdr`

header della parte dati

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

- message.h

3.6 Riferimenti per la struct message_hdr_t

header del messaggio

`#include <message.h>`

Campi

- op_t op
- char sender [MAX_NAME_LENGTH+1]

3.6.1 Descrizione dettagliata

header del messaggio

3.6.2 Documentazione dei campi

3.6.2.1 op

`op`

tipo di operazione richiesta al server

3.6.2.2 sender

`sender`

nickname del mittente

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

- message.h

3.7 Riferimenti per la struct message_t

tipo del messaggio

`#include <message.h>`

Campi

- message_hdr_t hdr
- message_data_t data

3.7.1 Descrizione dettagliata

tipo del messaggio

3.7.2 Documentazione dei campi

3.7.2.1 data

`data`

dati

3.7.2.2 hdr

`hdr`

header

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

- message.h

3.8 Riferimenti per la struct msgnode_t

Struttura che identifica un messaggio (msg + puntatore a next msg)

```
#include <msgqueue.h>
```

Campi

- message_t * msg
- struct msgnode_s * next

3.8.1 Descrizione dettagliata

Struttura che identifica un messaggio (msg + puntatore a next msg)

3.8.2 Documentazione dei campi

3.8.2.1 msg

`msg`

variabile di tipo message_t (contenente il messaggio da salvare)

3.8.2.2 next

`next`

puntatore al messaggio successivo

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

- msgqueue.h

3.9 Riferimenti per la struct msgqueue_t

Implementazione di una lista di messaggi di dimensione finita (historysize)

`#include <msgqueue.h>`

Campi

- size_t size
- size_t dim_max
- msgnode_t * head
- msgnode_t * tail

3.9.1 Descrizione dettagliata

Implementazione di una lista di messaggi di dimensione finita (historysize)

3.9.2 Documentazione dei campi

3.9.2.1 dim_max

`dim_max`

dimensione massima della history (historysize)

3.9.2.2 head

`head`

puntatore al messaggio in testa

3.9.2.3 size

`size`

dimensione corrente della lista (<=dim_max)

3.9.2.4 tail

`tail`

puntatore al messaggio in coda

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

- msgqueue.h

3.10 Riferimenti per la struct operation_t

Campi

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

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

- client.c

3.11 Riferimenti per la struct pool

struttura che implementa un pool di thread

`#include <threadlib.h>`

Campi

- int size
- pthread_t * thread

3.11.1 Descrizione dettagliata

struttura che implementa un pool di thread

3.11.2 Documentazione dei campi

3.11.2.1 size

`size`

numero di thread da voler creare nel pool

3.11.2.2 thread

`thread`

thread del pool

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

- threadlib.h

3.12 Riferimenti per la struct queue

Struttura che implemente una coda concorrente.

`#include <queuelib.h>`

Campi

- int front
- int rear
- int dim
- int * elem
- pthread_mutex_t mtx
- pthread_cond_t cnd1

3.12.1 Descrizione dettagliata

Struttura che implemente una coda concorrente.

La coda è strutturata in modo tale da essere usata in maniera concorrente fra più thread, quindi utilizza una variabile di mutua esclusione che ne garantisce la consistenza. Inoltre il comportamento della coda è modellato con la caratteristica di bloccare (sospendere) chi richiede di estrarre un elemento se essa è vuota, quindi utilizza una variabile di condizione utilizzata in modo da far sospende il thread che voglia estrarre un elemento dalla coda vuota, e svegliato dal primo che ne inserisce un elemento.

3.12.2 Documentazione dei campi

3.12.2.1 cnd1

`cnd1`

variabile di condizione per coda vuota

3.12.2.2 dim

`dim`

dimensione della coda

3.12.2.3 elem

`elem`

elemento della coda (fd client accodato)

3.12.2.4 front

`front`

primo elemento da estrarre nella coda

3.12.2.5 mtx

`mtx`

variabile di mutua esclusione per accesso concorrente

3.12.2.6 rear

`rear`

ultimo elemento inserito nella coda

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

- queuelib.h

3.13 Riferimenti per la struct statistics

`#include <stats.h>`

Campi

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

3.13.1 Descrizione dettagliata

Struttura che implementa un delle statistiche di utilità per il server

3.13.2 Documentazione dei campi

3.13.2.1 ndelivered

```
unsigned long ndelivered
```

n. di messaggi testuali consegnati

3.13.2.2 nerrors

```
unsigned long nerrors
```

n. di messaggi di errore

3.13.2.3 nfiledelivered

```
unsigned long nfiledelivered
```

n. di file consegnati

3.13.2.4 nfilenotdelivered

```
unsigned long nfilenotdelivered
```

n. di file non ancora consegnati

3.13.2.5 nnotdelivered

```
unsigned long nnotdelivered
```

n. di messaggi testuali non ancora consegnati

3.13.2.6 nonline

```
unsigned long nonline
```

n. di utenti connessi

3.13.2.7 nusers

```
unsigned long nusers
```

n. di utenti registrati

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

- stats.h

3.14 Riferimenti per la struct user_data_t

Struttura dati utilizzata come "value" per la 1° mappa utenti (users)

`#include <userlib.h>`

Campi

- char name [MAX_NAME_LENGTH+1]
- unsigned long fd
- msgqueue_t * msgq

3.14.1 Descrizione dettagliata

Struttura dati utilizzata come "value" per la 1° mappa utenti (users)

3.14.2 Documentazione dei campi

3.14.2.1 fd

`fd`

Relativo file descriptor

3.14.2.2 msgq

`msgq`

Coda messaggi ricevuti (history)

3.14.2.3 name

`name`

Nickname user registrato

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

- userlib.h

3.15 Riferimenti per la struct users_struct_t

Struttura utilizzata dal Server chatty, per memorizzare gli utenti che iscrivono e i loro messaggi.

`#include <userlib.h>`

Campi

- icl_hash_t * users
- icl_hash_t * fdusr
- pthread_mutex_t * mtx
- unsigned int historysize
- unsigned int **usersOnline**

3.15.1 Descrizione dettagliata

Struttura utilizzata dal Server chatty, per memorizzare gli utenti che iscrivono e i loro messaggi.

3.15.2 Documentazione dei campi

3.15.2.1 fdusr

`fdusr`

2°tabella hash usata per la memorizzazione dei file descriptor degli utenti online <key,data>=<fd(stringa),nickname>

3.15.2.2 historysize

`historysize`

Dimensione della History dei messaggi

3.15.2.3 mtx

`mtx`

Variabile di mutua esclusione usate per accedere alla struttura

3.15.2.4 users

`users`

1°tabella hash usata per la memorizzazione degli utenti registrati <key,data>=<nickname,user_data_t>

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

- userlib.h

4 Documentazione dei file

4.1 Riferimenti per il file chatty.c

Implementazione del Server chatty.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <sys/types.h>
#include <signal.h>
#include <pthread.h>
#include <errno.h>
#include <getopt.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <sys/un.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "connections.h"
#include "queuelib.h"
#include "threadlib.h"
#include "parser.h"
#include "icl_hash.h"
#include "userlib.h"
#include "stats.h"
```

Grafo delle dipendenze di inclusione per chatty.c:

Definizioni

- #define **_POSIX_C_SOURCE** 200809L

Funzioni

- void terminateServer ()

  Funzione di terminazione Server (SIGINT/SIGQUIT/SIGTERM)
- void plotStats ()

  Funzione che stampa le statistiche del Server (SIGUSR1)
- void signalHandler ()

  Funzione che cattura i segnali (TERMINAZIONE + SIGUSR1)
- int executeReq (int fd, message_t msg)

  Funzione usata dai Thread Worker per gestire le OP.
- void ∗ worker (void ∗arg)

  Funzione passata ai thread worker.
- int **main** (int argc, char ∗argv[ ])

Variabili

- conf_var * **config**
- queue * **coda**
- users_struct_t * **usr**
- fd_set **set**
- struct statistics **chattyStats** = {0,0,0,0,0,0,0}
- volatile sig_atomic_t **alive** =1

4.1.1 Descrizione dettagliata

Implementazione del Server chatty.

Chatty è un server che permette a dei client di poter chattare fra di loro, consentendogli di scambiarsi messaggi sia testuali che file. Il server è implementato in modo da riuscire a servire in maniera concorrente più client, attraverso un pool di thread che riescono a servire in maniera efficente i client e i messaggi scambiati. Il design adottato è quello di un server Master-Slave (Master-Worker), dove in questo caso, il thread main del processo chatty, lancia "n" Worker, e si mette in ascolto su un socket di tipo AF_UNIX per servire i client. Il main quindi accoglie i fd dei client da servire, e schedula i Worker per poter eseguire l'operazione richiesta dagli utenti della chat. I Worker cicleranno all'infinito estraendo da una coda i filedescriptor da servire, aspettando di essere terminati da uno dei segnali registrati dal main.

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.1.2 Documentazione delle funzioni

4.1.2.1 executeReq()

```
int executeReq (
            int fd,
            message_t msg )
```

Funzione usata dai Thread Worker per gestire le OP.

La funzione prende il messaggio del client, precedentemente letto dal thread worker e lo switcha fra le varie operazioni possibile, dando una risposta al client secondo le varie casistiche.

Parametri

fd	filedescriptor del client da servire
msg	il messaggio letto dal socket

Restituisce

0 in caso di successo (il Server è riuscito a gestire la richiesta del client)
-1 in caso di fallimento

4.1.2.2 worker()

```
void* worker (
            void * arg )
```

Funzione passata ai thread worker.

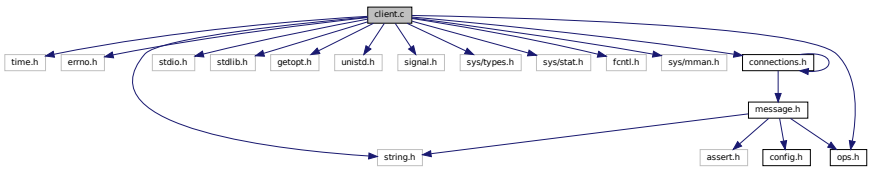
La funzione esegue un ciclo infinito (finchè non viene interrotto da uno dei segnali mascherati) dentro il quale estrae da una coda concorrente i file descriptor dei client che devono essere serviti. Appena estratto, legge dal socket l'intero messaggio e lo passa alla funzione executeReq che lo processa. In base all'esito della funzione decide se rimettere il tutto dentro al set dei fd da servire, oppure di disconnettere il client

4.2 Riferimenti per il file client.c

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>
```

Grafo delle dipendenze di inclusione per client.c:



Strutture dati

- struct operation_t

Definizioni

- #define **_POSIX_C_SOURCE** 200809L

Funzioni

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

4.2.1 Descrizione dettagliata

Semplice client di test.

4.3 Riferimenti per il file config.h

File contenente alcune define con valori massimi utilizzabili.

Definizioni

- #define **MAX_NAME_LENGTH** 32

Ridefinizioni di tipo (typedef)

- typedef int **make_iso_compilers_happy**

4.3.1 Descrizione dettagliata

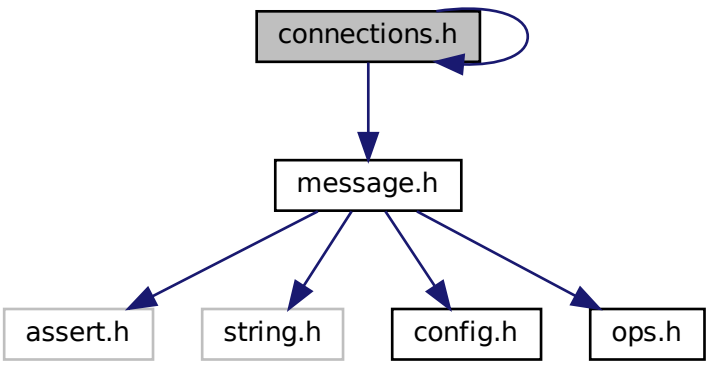
File contenente alcune define con valori massimi utilizzabili.

4.4 Riferimenti per il file connections.h

Contine le funzioni che implementano il protocllo di comunicazione tra il client ed il server.

```
#include "message.h"
#include "connections.h"
```

Grafo delle dipendenze di inclusione per connections.h:



Definizioni

- #define **MAX_RETRIES** 10
- #define **MAX_SLEEPING** 3
- #define **UNIX_PATH_MAX** 64

Funzioni

- int openConnection (char ∗path, unsigned int ntimes, unsigned int secs)

  Apre una connessione attravverso un socket AF_UNIX verso il server.
- int readHeader (long connfd, message_hdr_t ∗hdr)

  Legge l'header del messaggio dal socket.
- int readData (long fd, message_data_t ∗data)

  Legge il body del messaggio dal socket.
- int readMsg (long fd, message_t ∗msg)

  Legge l'intero messaggio (header + data) dal socket.
- int sendRequest (long fd, message_t ∗msg)

  Scrive un intero messaggio sul socket.
- int sendData (long fd, message_data_t ∗msg)

  Scrive il body del messaggio sul socket.
- int sendHeader (long fd, message_hdr_t ∗hdr)

  Scrive l'header del messaggio sul socket.

4.4.1 Descrizione dettagliata

Contine le funzioni che implementano il protocllo di comunicazione tra il client ed il server.

Connections implementa il core della comunicazione tra client e server. È in grado di leggere e scrivere header e data di messaggi di tipo message_t. La comunicazione gira su un socket di tipo AF_UNIX.

Si veda anche

message.h

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.4.2 Documentazione delle funzioni

4.4.2.1 openConnection()

```
int openConnection (
            char * path,
            unsigned int ntimes,
            unsigned int secs )
```

Apre una connessione attravverso un socket AF_UNIX verso il server.

Parametri

path	Path del socket AF_UNIX
ntimes	numero massimo di tentativi di retry
secs	tempo di attesa tra due retry consecutive

Restituisce

fd associato alla connessione in caso di successo
-1 in caso di errore

4.4.2.2 readData()

```
int readData (
            long fd,
            message_data_t * data )
```

Legge il body del messaggio dal socket.

Parametri

fd	descrittore della connessione
data	puntatore al body del messaggio

Restituisce

<=0 se c'e' stato un errore (se <0 errno deve essere settato, se == 0 connessione chiusa)
1 in caso di successo

4.4.2.3 readHeader()

```
int readHeader (
            long connfd,
            message_hdr_t * hdr )
```

Legge l'header del messaggio dal socket.

Parametri

fd	descrittore della connessione
hdr	puntatore all'header del messaggio da ricevere

Restituisce

<=0 se c'e' stato un errore (se <0 errno deve essere settato, se == 0 connessione chiusa)
1 in caso di successo

4.4.2.4 readMsg()

```
int readMsg (
            long fd,
            message_t * msg )
```

Legge l'intero messaggio (header + data) dal socket.

Parametri

fd	descrittore della connessione
data	puntatore al messaggio

Restituisce

<=0 se c'e' stato un errore (se <0 errno deve essere settato, se == 0 connessione chiusa)
1 in caso di successo

4.4.2.5 sendData()

```
int sendData (
            long fd,
            message_data_t * msg )
```

Scrive il body del messaggio sul socket.

Parametri

fd	descrittore della connessione
msg	puntatore al messaggio da inviare

Restituisce

<=0 se c'e' stato un errore
1 in caso di successo

4.4.2.6 sendHeader()

```
int sendHeader (
            long fd,
            message_hdr_t * hdr )
```

Scrive l'header del messaggio sul socket.

Parametri

fd	descrittore della connessione
hdr	puntatore all'header da inviare

Restituisce

<=0 se c'è stato un errore
1 in caso di successo

4.4.2.7 sendRequest()

```
int sendRequest (
            long fd,
            message_t * msg )
```

Scrive un intero messaggio sul socket.

Parametri

fd	descrittore della connessione
msg	puntatore al messaggio da inviare

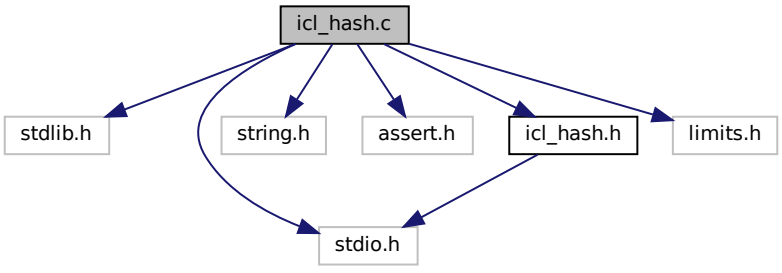
Restituisce

<=0 se c'e' stato un errore
1 in caso di successo

4.5 Riferimenti per il file icl_hash.c

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

Grafo delle dipendenze di inclusione per icl_hash.c:



Definizioni

- #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))

Funzioni

- 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)

4.5.1 Descrizione dettagliata

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 :-)

Autore

Jakub Kurzak

4.5.2 Documentazione delle funzioni

4.5.2.1 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.

Parametri

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

Restituisce

pointer to new hash table.

4.5.2.2 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).

Parametri

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

Restituisce

0 on success, -1 on failure.

4.5.2.3 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).

Parametri

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

Restituisce

0 on success, -1 on failure.

4.5.2.4 icl_hash_dump()

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

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

Parametri

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

Restituisce

0 on success, -1 on failure.

4.5.2.5 icl_hash_find()

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

Search for an entry in a hash table.

Parametri

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

Restituisce

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

4.5.2.6 icl_hash_insert()

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

Insert an item into the hash table.

Parametri

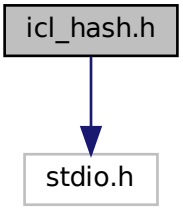
ht	– the hash table
key	– the key of the new item
data	– pointer to the new item's data

Restituisce

pointer to the new item. Returns NULL on error.

4.6 Riferimenti per il file icl_hash.h

`#include <stdio.h>`
Grafo delle dipendenze di inclusione per icl_hash.h:



Strutture dati

- struct icl_entry_t
- struct icl_hash_t

Definizioni

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

Funzioni

- 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 *))

4.6.1 Descrizione dettagliata

Header file for icl_hash routines.

4.6.2 Documentazione delle definizioni

4.6.2.1 icl_hash_foreach

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

Valore:

```
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)
```

4.6.3 Documentazione delle funzioni

4.6.3.1 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.

Parametri

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

Restituisce

pointer to new hash table.

4.6.3.2 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).

Parametri

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

Restituisce

0 on success, -1 on failure.

4.6.3.3 icl_hash_dump()

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

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

Parametri

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

Restituisce

0 on success, -1 on failure.

4.6.3.4 icl_hash_find()

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

Search for an entry in a hash table.

Parametri

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

Restituisce

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

4.6.3.5 icl_hash_insert()

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

Insert an item into the hash table.

Parametri

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

Restituisce

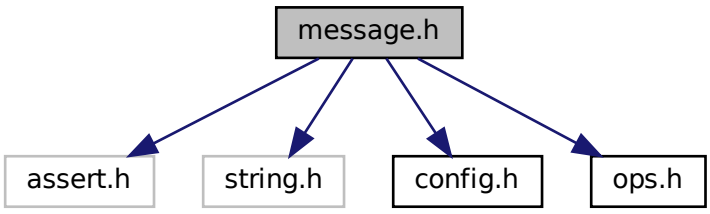
pointer to the new item. Returns NULL on error.

4.7 Riferimenti per il file message.h

Contiene le strutture dei messaggi + alcune funzioni per manipolarli.

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

Grafo delle dipendenze di inclusione per message.h:



Strutture dati

- struct message_hdr_t
  header del messaggio
- struct message_data_hdr_t
  header della parte dati
- struct message_data_t
  body del messaggio
- struct message_t
  tipo del messaggio

4.7.1 Descrizione dettagliata

Contiene le strutture dei messaggi + alcune funzioni per manipolarli.

Autore

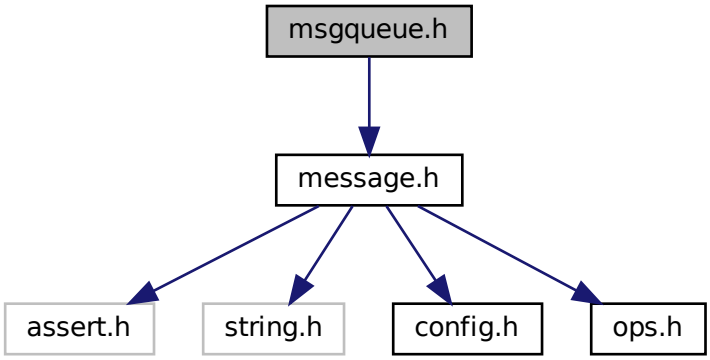
Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.8 Riferimenti per il file msgqueue.h

Implementazione della history messaggi.

```
#include "message.h"
```

Grafo delle dipendenze di inclusione per msgqueue.h:



Strutture dati

- struct msgnode_t

  Struttura che identifica un messaggio (msg + puntatore a next msg)
- struct msgqueue_t

  Implementazione di una lista di messaggi di dimensione finita (historysize)

Funzioni

- msgqueue_t ∗ createMsgQueue (int dim)

  Crea una coda di messaggi (history) di dimensione dim (historysize)
- msgnode_t ∗ createMsgNode ()

  Crea un elemento per la history (di tipo: msgnode_t) facendone una copia completa.
- message_t ∗ popMsgQueue (msgqueue_t ∗queue)

  Estrae il primo messaggio dalla hsitory.
- int pushMsgQueue (msgqueue_t ∗queue, message_t ∗msg)

  Inserisce a fine coda il messaggio.
- void destroyMsgQueue (msgqueue_t ∗queue)

  Funzione che dealloca dalla memoria tutta la history.
- void destroyMsgNode (msgnode_t ∗node)

  Distrugge un nodo della history.

4.8.1 Descrizione dettagliata

Implementazione della history messaggi.

Msqueue implementa una coda di messaggi, utilizzata dal server chatty per fare lo store dei messaggi ricevuti fino a un max di "historysize".

Si veda anche

message.h

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.8.2 Documentazione delle funzioni

4.8.2.1 createMsgNode()

```
msgnode_t* createMsgNode ( )
```

Crea un elemento per la history (di tipo: msgnode_t) facendone una copia completa.

Parametri

msgtocpy	messaggio che deve essere copiato nella history (message_t)

Restituisce

ritorna il nodo della lista contenente il messaggio copiato

4.8.2.2 createMsgQueue()

```
msgqueue_t* createMsgQueue (
            int dim )
```

Crea una coda di messaggi (history) di dimensione dim (historysize)

Parametri

dim	dimensione massima coda

Restituisce

ritorna la coda messaggi (msgqueue_t)

Crea una coda di messaggi (history) di dimensione dim (historysize)

Msqueue implementa una coda di messaggi, utilizzata dal server chatty per fare lo store dei messaggi ricevuti fino a un max di "historysize".

Si veda anche

message.h

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore Crea una coda di messaggi (history) di dimensione dim (historysize)

Parametri

dim	dimensione massima coda

Restituisce

ritorna la coda messaggi (msgqueue_t)

4.8.2.3 destroyMsgNode()

```
void destroyMsgNode (
            msgnode_t * node )
```

Distrugge un nodo della history.

destroyMsgNode

Parametri

node	nodo da deallocare (msgnode_t)

4.8.2.4 destroyMsgQueue()

```
void destroyMsgQueue (
            msgqueue_t * queue )
```

Funzione che dealloca dalla memoria tutta la history.

Parametri

queue	coda dei messaggi da eliminare

4.8.2.5 popMsgQueue()

```
message_t* popMsgQueue (
            msgqueue_t * queue )
```

Estrae il primo messaggio dalla hsitory.

Parametri

queue	coda da dove estrarre il messaggio

Restituisce

ritorno il messaggio estratto
(NULL se non ci sono messaggi)

4.8.2.6 pushMsgQueue()

```
int pushMsgQueue (
            msgqueue_t * queue,
            message_t * msgtocpy )
```

Inserisce a fine coda il messaggio.

Parametri

queue	history dove inserire i messaggi
msgtocpy	il messaggio che deve essere copiato nella history

Restituisce

-1 in caso di fallimento (queueu==NULL || msgtocpy ==NULL)
0 in caso di successo

4.9 Riferimenti per il file ops.h

File che definisce le operazioni effettuabili da un client e i tipi di messaggi di risposta tra client/server (e viceversa)

Tipi enumerati (enum)

- 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, CREATEGROUP_OP = 10, ADDGROUP_OP = 11,
  DELGROUP_OP = 12, 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 }

4.9.1 Descrizione dettagliata

File che definisce le operazioni effettuabili da un client e i tipi di messaggi di risposta tra client/server (e viceversa)

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.9.2 Documentazione dei tipi enumerati

4.9.2.1 op_t

`enum op_t`

Valori del tipo enumerato

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
CREATEGROUP_OP	richiesta di disconnessione
ADDGROUP_OP	richiesta di creazione di un gruppo
DELGROUP_OP	richiesta di aggiunta ad un gruppo
OP_OK	richiesta di rimozione da un gruppo

4.10 Riferimenti per il file parser.h

Implementazione di una struttura per le variabili di configurazione.

Strutture dati

- struct conf_var

  struttura per memorizzare variabili di configurazione

Funzioni

- void trova_val (char ∗buf, int i, int type, int scanned)

  Funzione di supporto a "parse" che serve a estrapolare sottostringhe.
- conf_var ∗ parse (char ∗conffile)

  Estrapola le variabili di configurazione del server parsandole da file.

4.10.1 Descrizione dettagliata

Implementazione di una struttura per le variabili di configurazione.

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

- funzione per parsare da file i valori

4.10.2 Documentazione delle funzioni

4.10.2.1 parse()

```
conf_var* parse (
            char * conffile )
```

Estrapola le variabili di configurazione del server parsandole da file.

Parametri

conffile	

Restituisce

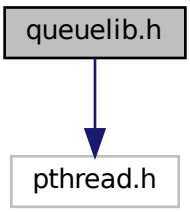
conf_var (struttura contenente le variabili)

4.11 Riferimenti per il file queuelib.h

Implementazione di una coda che viene acceduta in maniera concorrente.

```
#include <pthread.h>
```

Grafo delle dipendenze di inclusione per queuelib.h:



Strutture dati

- struct queue

  Struttura che implemente una coda concorrente.

Definizioni

- #define **QUEUELIB_H_**
- #define **KILLTHREAD** 99999999

Funzioni

- int isFull (queue ∗coda)
- int isEmpty (queue ∗coda)
- queue ∗ createQueue (int dim)

  Crea una coda concorrente.
- int enQueue (queue ∗coda, int elem)

  Funzione che accoda un elemento alla Coda.
- int deQueue (queue ∗coda)

  Funzione che estrae un elemento dalla Coda.
- void destroyQueue (queue ∗coda)

  libera la memoria da tutte le strutture utilizzate dalla coda

4.11.1 Descrizione dettagliata

Implementazione di una coda che viene acceduta in maniera concorrente.

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.11.2 Documentazione delle funzioni

4.11.2.1 createQueue()

```
queue* createQueue (
            int dim )
```

Crea una coda concorrente.

Parametri

dim	dimensione coda

Restituisce

la coda concorrente

4.11.2.2 deQueue()

```
int deQueue (
            queue * coda )
```

Funzione che estrae un elemento dalla Coda.

La funzione deQueue, estrae un elemento per conto del chiamante e lo restituisce se ovviamente la coda è non vuota. In caso di coda vuota, si sospende autonomamente sulla variabile di condizionamento cnd1, in attesa di essere risvegliato da qualche produttore. Caso di riguardo è quando estrae il messaggio KILLTHREAD, esso indica che il chiamante di deQueue, deve terminare.

Parametri

coda	coda da gestire

Restituisce

elemento in testa alla coda (si sospende se è vuota)
-1 se c'è qualche errore

4.11.2.3 destroyQueue()

```
void destroyQueue (
            queue * coda )
```

libera la memoria da tutte le strutture utilizzate dalla coda

Parametri

coda	coda da eliminare

4.11.2.4 enQueue()

```
int enQueue (
            queue * coda,
            int elem )
```

Funzione che accoda un elemento alla Coda.

La procedura di enQueue, inserisce un elemento in coda se e solo se essa è non piena e non appena termina l'inserimento, prima di rilasciare la lock, segnala un eventuale thread che si è sospeso in attesa di estrarre.

Parametri

coda	coda da gestire
elem	elemento da accodare

Restituisce

0 se accoda a buon fine
-1 se c'è un errore

4.11.2.5 isEmpty()

```
int isEmpty (
            queue * coda )
```

Parametri

coda	

Restituisce

true se la coda è vuota, false se la coda è non vuota

4.11.2.6 isFull()

```
int isFull (
            queue * coda )
```

Parametri

coda	

Restituisce

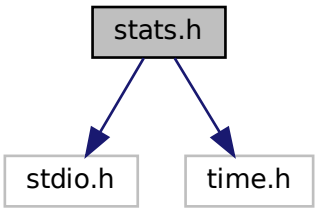
true se la coda è piena, false se la coda è non piena

4.12 Riferimenti per il file stats.h

Contiene funzioni di utilità per le statistiche.

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

Grafo delle dipendenze di inclusione per stats.h:



Strutture dati

- struct statistics

4.12.1 Descrizione dettagliata

Contiene funzioni di utilità per le statistiche.

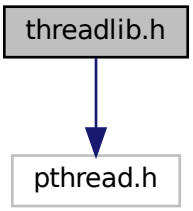
Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.13 Riferimenti per il file threadlib.h

Libreria che implementa un pool di thread.

`#include <pthread.h>`
Grafo delle dipendenze di inclusione per threadlib.h:



Strutture dati

- struct pool

  struttura che implementa un pool di thread

Funzioni

- pool ∗ createPool (int numt)

  crea un pool di numt threads
- void initPool (pool ∗pool, void ∗(∗start_routine)(void ∗))

  Inizializza un pool di thread con un task (routine)
- void destroyPool (pool ∗pool)

  Ripulisce le strutture utilizzate dal pool e il pool stesso.

4.13.1 Descrizione dettagliata

Libreria che implementa un pool di thread.

Threadlib implementa un pool di thread secondo il modello Master + Worker. Nella libreria inoltre vengono implementate delle funzioni per istanziare ed avviare i thread con dei task e una funzione per eseguire una cleanup.

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.13.2 Documentazione delle funzioni

4.13.2.1 createPool()

```
pool* createPool (
            int numt )
```

crea un pool di numt threads

Parametri

numt	numero di thread worker che si vogliono creare

Restituisce

pool ritorna un pool di numt threads

crea un pool di numt threads

Threadlib implementa un pool di thread secondo il modello Master + Worker. Nella libreria inoltre vengono implementate delle funzioni per istanziare ed avviare i thread con dei task e una funzione per eseguire una cleanup.

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore crea un pool di numt threads

Parametri

numt	numero di thread worker che si vogliono creare

Restituisce

pool ritorna un pool di numt threads

4.13.2.2 destroyPool()

```
void destroyPool (
            pool * pool )
```

Ripulisce le strutture utilizzate dal pool e il pool stesso.

Parametri

pool	il pool di thread da deallocare

4.13.2.3 initPool()

```
void initPool (
            pool * pool,
            void *(*)(void *) start_routine )
```

Inizializza un pool di thread con un task (routine)

Parametri

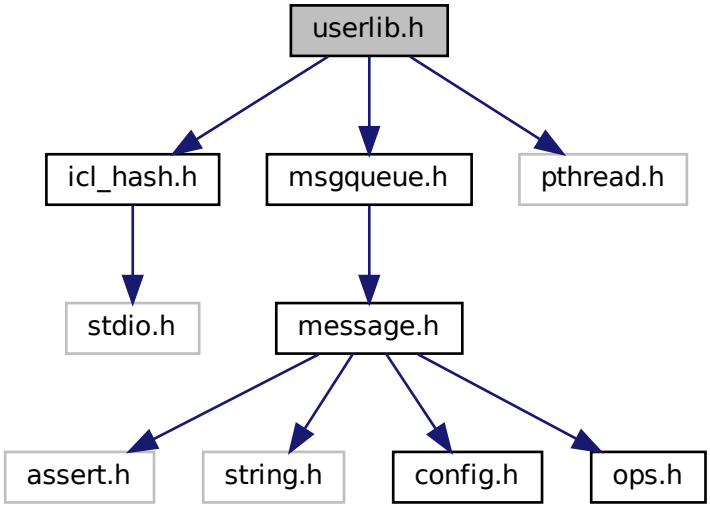
pool	il pool di thread da voler inizializzare
start_routine	il task da voler passare ai threads

4.14 Riferimenti per il file userlib.h

Implementazione registrazione utenti + Implementazione operazione utenti.

```
#include "icl_hash.h"
#include "msgqueue.h"
#include <pthread.h>
```

Grafo delle dipendenze di inclusione per userlib.h:



Strutture dati

- struct users_struct_t

  Struttura utilizzata dal Server chatty, per memorizzare gli utenti che iscrivono e i loro messaggi.
- struct user_data_t

  Struttura dati utilizzata come "value" per la 1°mappa utenti (users)

Funzioni

- users_struct_t * createUsersStruct (unsigned long history, unsigned long nbuckets)

  Crea la struttura principale per memorizzare gli utenti.
- int registerUser (users_struct_t *tab, char *nick, unsigned long fd)

  Funzione che registra gli utenti nell'apposita struttura dati.

- int connectUser (users_struct_t *tab, char *nick, unsigned long fd)

  Connette l'utente che ne fa richiesta aggiornandone il relativo filedescriptor.
- int unregisterUser (users_struct_t *tab, char *nick, int fd)

  Deregistra l'utente che ne fa richiesta.
- int disconnectUser (users_struct_t *tab, char *nick, unsigned long fd)

  Disconnette l'utente che ne fa richiesta.
- int getOnlineList (users_struct_t *tab, char **list)

  Funzione che inizializza list con i nickname degli utenti attualmente online.
- msgqueue_t * getHistory (users_struct_t *tab, char *nick)

  Funzione che restituisce una copia dei messaggi in coda.
- int getUserFD (users_struct_t *tab, char *nick)

  Funzione che restituisce il file descriptor associato all'utente "nick".
- int getAllUsersFD (users_struct_t *tab, char *nick, int **fds)

  Funzione che riempie il vettore di interi fds con i fd degli user online.
- int postOnHistory (users_struct_t *tab, message_t *msg)

  Funzione che posta un msg nella history di msg->receiver.
- int postOnHistoryAll (users_struct_t *tab, message_t *msg)

  Funzione che posta un msg nella history di tutti gli utenti online.
- void free_data (void *arg)

  Distrugge le strutture dati relative alla memorizzazione utenti.
- void destroyUsersStruct (users_struct_t *tab)

  Distrugge le strutture dati relative alla memorizzazione utenti.

4.14.1 Descrizione dettagliata

Implementazione registrazione utenti + Implementazione operazione utenti.

Userlib è il core di tutta la memorizzazione degli utenti riguardo le principali operazioni di: registrazione/deregistrazione login/disconnessione, e della memorizzazione dei messaggi/file inviati, si serve di due strutture hash: <users> e <fdusr> nei quali si memorizzano i nicname degli user e i relativi filedescriptor nel momento in cui emettono richieste al serve, inoltre sono memorizzate alcune info di utilità per il runtime.

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore

4.14.2 Documentazione delle funzioni

4.14.2.1 connectUser()

```
int connectUser (
            users_struct_t * tab,
            char * nick,
            unsigned long fd )
```

Connette l'utente che ne fa richiesta aggiornandone il relativo filedescriptor.

Parametri

tab	struttura dove è registrato l'utente
nick	username utente che vuole essere connesso
fd	filedescriptor utente che lo richiede

Restituisce

-1 Se l'utente non è registrato
-2 Se l'utente è già connesso
0 Se va a buon fine

già collegato

4.14.2.2 createUsersStruct()

```
users_struct_t* createUsersStruct (
            unsigned long historysize,
            unsigned long nbuckets )
```

Crea la struttura principale per memorizzare gli utenti.

Per la memorizzazione di un utente si utilizzano due strutture hash: 1) per memorizzare la stringa dell'utente con le relative informazione di utilità e 2)per memorizzare il file descriptor dell'user che richiede di loggarsi associandolo con il relativo nickname. Il motivo di tale scelta implementativa è che il client potrebbe disconnettersi in "maniera implicita" rendendo la disconnessione possibile soltanto per mezzo del suo file descriptor.

Parametri

historysize	dimensione massima coda messaggi ricevuti
nbuckets	dimensione inziale tabella hash utenti

Restituisce

puntatore a users_struct_t

4.14.2.3 destroyUsersStruct()

```
void destroyUsersStruct (
            users_struct_t * tab )
```

Distrugge le strutture dati relative alla memorizzazione utenti.

Parametri

tab	tabella utenti

4.14.2.4 disconnectUser()

```
int disconnectUser (
            users_struct_t * tab,
            char * nick,
            unsigned long fd )
```

Disconnette l'utente che ne fa richiesta.

La disconnessione può essere di due tipo 1)Implicita e 2)Esplcita. Ciò per come è strutturato il client. Infatti, il client potrebbe uscire in un qualsiasi momento del suo ciclo di vita, questo perchè non è detto che intenda mandare esplicitamente al server un messaggio di disconnessione. Perciò il metodo di disconnessione è implementato in modo tale che prende come paramentri sia l'username che il filedescriptor e qual'ora l'username passato sia NULL, la disconnessione verrà trattata in modo implicito.

Parametri

tab	struttura dove è registrato l'utente
nick	username utente che vuole essere disconnesso

Restituisce

-1 Se l'utente non è online
-2 Se l'utente non è registrato
0 Se va a buon fine

4.14.2.5 free_data()

```
void free_data (
            void * arg )
```

Distrugge le strutture dati relative alla memorizzazione utenti.

Parametri

tab	tabella utenti

Distrugge le strutture dati relative alla memorizzazione utenti.

Userlib è il core di tutta la memorizzazione degli utenti riguardo le principali operazioni di: registrazione/deregistrazione login/disconnessione, e della memorizzazione dei messaggi/file inviati, si serve di due strutture hash: <users> e <fdusr> nei quali si memorizzano i nicname degli user e i relativi filedescriptor nel momento in cui emettono richieste al server, inoltre sono memorizzate alcune info di utilità per il runtime.

Autore

Stefano Spadola 534919 Si dichiara che il contenuto di questo file e' in ogni sua parte opera originale dell'autore Funzione di clean up, elimina il campo

della 1° tabella hash utenti (users)

4.14.2.6 getAllUsersFD()

```
int getAllUsersFD (
            users_struct_t * tab,
            char * nick,
            int ** fds )
```

Funzione che riempie il vettore di interi fds con i fd degli user online.

Parametri

tab	struttura dove è registrato l'utente
nick	username dell'Utente che ne fa richiesta
fds	puntatore a vettore di filedescriptor

Restituisce

-1 on failure
#user online on success

4.14.2.7 getHistory()

```
msgqueue_t* getHistory (
            users_struct_t * tab,
            char * nick )
```

Funzione che restituisce una copia dei messaggi in coda.

Questa funzione ritorna una copia dei messaggi nella History Si preferisce farne una copia intera per motivi di consistenza, in quanto un altro thread del server, potrebbe voler registrare un messaggio mandato da un altro utente.

Parametri

tab	struttura dove è registrato l'utente
nick	username dell'Utente

Restituisce

NULL se l'utente è offline
msgqueue contenente la history di "nick"

4.14.2.8 getOnlineList()

```
int getOnlineList (
            users_struct_t * tab,
            char ** list )
```

Funzione che inizializza list con i nickname degli utenti attualmente online.

Parametri

tab	struttura dove è registrato l'utente
list	puntatore ad array di caratteri

Restituisce

-1 Se c'è un errore
>=0 #utenti online

4.14.2.9 getUserFD()

```
int getUserFD (
            users_struct_t * tab,
            char * nick )
```

Funzione che restituisce il file descriptor associato all'utente "nick".

Parametri

tab	struttura dove è registrato l'utente
nick	username dell'Utente

Restituisce

-1 se nick non esiste
0 se nick non è online
fd di nick

4.14.2.10 postOnHistory()

```
int postOnHistory (
            users_struct_t * tab,
            message_t * msg )
```

Funzione che posta un msg nella history di msg->receiver.

Parametri

tab	struttura dove è registrato l'utente
msg	messaggio da postare

Restituisce

-1 on failure
0 in caso di successo

4.14.2.11 postOnHistoryAll()

```
int postOnHistoryAll (
            users_struct_t * tab,
            message_t * msg )
```

Funzione che posta un msg nella history di tutti gli utenti online.

Parametri

tab	struttura dove è registrato l'utente
msg	messaggio da postare

Restituisce

-1 on failure
0 in caso di successo

4.14.2.12 registerUser()

```
int registerUser (
            users_struct_t * tab,
            char * nick,
            unsigned long fd )
```

Funzione che registra gli utenti nell'apposita struttura dati.

Parametri

tab	struttura dove registrare l'utente
nick	username utente che vuole essere registrato

Restituisce

-1 se l'utente è già registrato
0 se va a buon fine

Funzione che registra gli utenti nell'apposita struttura dati.

Parametri

tab	struttura dove registrare l'utente
nick	username utente che vuole essere registrato
fd	filedescriptor utente che vuole essere connesso

Restituisce

-2 se c'è un errore di inserimento utente
-1 se l'utente è già registrato
0 se va a buon fine

4.14.2.13 unregisterUser()

```
int unregisterUser (
            users_struct_t * tab,
            char * nick,
            int fd )
```

Deregistra l'utente che ne fa richiesta.

Parametri

tab	struttura dove è registrato l'utente
nick	username utente che vuole essere deregistrato

Restituisce

-1 Se l'utente non è registrato
0 Se va a buon fine

Deregistra l'utente che ne fa richiesta.

Parametri

tab	struttura dove è registrato l'utente
nick	username utente che vuole essere deregistrato
fd	filedescriptor utente che vuole deregistrarsi

Restituisce

-1 Se l'utente non è registrato
0 Se va a buon fine

Indice analitico