# Bacheca Elettronica Remota

Realizzazione di una bacheca elettronica che permetta agli utenti autorizzati di inviare messaggi, i quali possano essere letti e/o aggiornati da qualsiasi utente che voglia accedere ad essa.

## Indice

1. Specifiche funzionali e di progetto 1
2. Descrizione generale dei file 2
   2.1 Server.c 2
   2.2 Client.c 2
   2.3 Utility.h 3
3. Flusso dell'interazione client/server 3
   3.1 Autenticazione/registrazione 3
   3.2 Lettura della bacheca 4
   3.3 Inserimento di un nuovo messaggio 5
   3.4 Cancellazione di un messaggio 6
   3.5 Chiusura della connessione 7
4. Gestione dei segnali e degli errori delle funzioni di sistema 7
5. Istruzioni per l'avviamento 8

## 1. Specifiche funzionali e di progetto

Il servizio permette ad un utente di accedere ad una bacheca elettronica previa autenticazione o registrazione. Una volta effettuato l'accesso, è possibile:

- leggere tutti i messaggi presenti in bacheca;
- aggiungere un nuovo messaggio;
- cancellare un messaggio (solo se quest'ultimo è stato aggiunto dall'utente che ne richiede la rimozione).

Il server multi-thread accetta e processa in concorrenza le richieste dei client, residenti anche su macchine diverse dal server.

Vengono impiegati due file:

- *users.txt* contiene le credenziali degli utenti registrati al servizio;
- *board.txt* contiene tutti i messaggi della bacheca.

Ogni messaggio contiene i campi:

- mittente;
- oggetto;

- testo.

# 2. Descrizione generali dei file

## 2.1 Server.c

Nella funzione `main`, il server si predispone alla connessione e stampa gli IP delle interfacce di rete affinché il client possa connettervisi. Dopodiché, apre il file contenente le credenziali degli utenti, inizializza i semafori e gestisce i segnali (descrizione al paragrafo 4).
Si mette, poi, in attesa dei client con un ciclo infinito in cui accetta una connessione (inviando anche un messaggio di avvenuta accettazione), crea un thread e affida a quest'ultimo la gestione delle richieste da parte del client. Il thread esegue la funzione `conn_handler` che si occupa, mediante un ciclo infinito, di ricevere e riconoscere la stringa comando ricevuta dal client (per leggere la bacheca, inserire, cancellare un messaggio o chiudere la connessione) e di assegnare, poi, la giusta funzione ad ogni comando ricevuto.

La terminazione del server è affidata alla funzione `server_close_handler` (descritta nel paragrafo 4).

## 2.2 Client.c

All'atto dell'avvio, sulla riga di comando, prende come parametro opzionale l'IP del server a cui connettersi (se omesso, verrà usato *127.0.0.1*).

Nella funzione `main`, il client si predispone alla connessione, attende il messaggio di accettazione (che confronta per rilevare eventuali errori) e gestisce i segnali (descrizione al paragrafo 4).
In un primo momento, si occupa dell'autenticazione/registrazione del client al servizio (si veda sotto); in un secondo, della stampa a video del menù dei comandi disponibili e dell'assegnazione dei relativi gestori in un ciclo infinito. Non sono ammessi comandi diversi da quelli stampati, per cui, in caso di immissione diversa, l'utente viene invitato ad inserire solo quelli elencati.

Una variabile globale tiene traccia dello username dell'utente corrente (utile per l'inserimento e la cancellazione dei messaggi).

Nel caso in cui più client vogliano accedere in scrittura ai file degli utenti o della bacheca contemporaneamente, sono stati impiegati dei semafori con nome per gestire l'accesso ai suddetti file.

La terminazione del client è affidata alla funzione `client_close` (descritta nel paragrafo 4).

## 2.3 Utility.h

È un file header incluso in entrambi i sorgenti, che contiene:

- le inclusioni delle librerie di sistema;
- le macro;
- un'utility di gestione dell'errore "`ERROR_HELPER`", che stampa un messaggio di errore con il relativo codice e fa terminare il chiamante;
- una funzione ausiliaria che crea named semaphores;
- una funzione ausiliaria che svuota le stringhe;
- una funzione ausiliaria che assegna le macro (stringhe) ai buffer.

# 3. Flusso dell'interazione client/server

## 3.1 Autenticazione/registrazione

Il client interroga l'utente per sapere se sia registrato o meno. In base alla risposta, chiama le funzioni `login` o `register_user`.

Nella fase di **login** (risposta: N/n), il client permette solo 3 tentativi, superati i quali avverrà la sua terminazione.

| Client | Server |
|---|---|
| 1) Invia la stringa comando "0" al server per avvertirlo che l'utente vuole autenticarsi. | |
| | 2) Risponde con la stringa "Ok". |
| 3) Controlla di aver ricevuto "Ok" da parte del server[1]. | |
| 4) Acquisisce le credenziali.<br>Le invia al server.<br>Salva lo username nella variabile globale. | |
| | 5) Controlla che la coppia username e password corrisponda a una di quelle contenute nel file delle credenziali.<br>Invia "Ok" se c'è il match e ritorna. |
| 6) Riceve l'"Ok", informa il client che l'autenticazione è avvenuta con successo[2] e ritorna. | |
| **Note/Gestione degli errori**<br>[1]: se riceve una stringa diversa da "Ok" (ci sono stati problemi durante l'invio/ricezione), viene chiamata `ERROR_HELPER`.<br>[2]: nel caso in cui riceva la stringa "Kk", vuol dire che non esiste alcun utente con lo username inserito e chiede all'utente se vuole registrarsi; se riceve | |

| una stringa diversa da “Ok”, viene chiamata `ERROR_HELPER`. |
|---|

Nella fase di **registrazione** (risposta: Y/y), il server controlla che non vi siano già credenziali con lo username ricevuto: in caso affermativo, informa il client che chiede all’utente di inserire un nickname diverso.

| Client | Server |
|---|---|
| 1) Invia la stringa comando “5” al server per avvertirlo che l’utente vuole registrarsi. | |
| | 2) Risponde con la stringa “Ok”. |
| 3) Controlla di aver ricevuto “Ok” da parte del server[1]. | |
| 4) Acquisisce le credenziali.<br>Le invia al server.<br>Salva lo username nella variabile globale. | |
| | 5) Controlla che lo username ricevuto non esista già.<br>Aggiunge le credenziali al file.<br>Manda una stringa “Ok” e ritorna. |
| 6) Riceve l’“Ok”, informa il client che la registrazione è avvenuta con successo[2] e ritorna. | |
| **Note/Gestione degli errori**<br>[1]: se riceve una stringa diversa da “Ok” (ci sono stati problemi durante l’invio/ricezione), viene chiamata `ERROR_HELPER`.<br>[2]: nel caso in cui riceva la stringa “Kk”, esiste già un utente con lo username inserito, per cui si torna al punto 4; se riceve una stringa diversa da “Ok”, viene chiamata `ERROR_HELPER`. | |

### 3.2 Lettura della bacheca

| Client | Server |
|---|---|
| 1) Invia la stringa comando “1” al server per avvertirlo che l’utente vuole leggere la bacheca. | |
| | 2) Risponde con la stringa “Ok”. |
| 3) Controlla di aver ricevuto “Ok” da parte del server[1]. | |
| | 4) Legge il numero delle righe della bacheca e le invia al client. |
| 5) Riceve il numero e lo salva. | |

| | |
|---|---|
| | 6) Invia le righe della bacheca e ritorna[2]. |
| 7) Stampa le righe della bacheca e ritorna[2]. | |
| **Note/Gestione degli errori**<br>[1]: se riceve una stringa diversa da "Ok" (ci sono stati problemi durante l'invio/ricezione), viene chiamata `ERROR_HELPER`.<br>[2]: queste azioni avvengono in sequenza per ogni riga della bacheca: lettura, invio, ricezione e stampa. | |

### 3.3 Inserimento di un nuovo messaggio

| Client | Server |
|---|---|
| 1) Invia la stringa comando "2" al server per avvertirlo che l'utente vuole aggiungere un messaggio. | |
| | 2) Risponde con la stringa "Ok". |
| 3) Controlla di aver ricevuto "Ok" da parte del server[1]. | |
| 4) Invia il proprio username al server. | |
| | 5) Riceve lo username e lo elabora<br>Manda un "Ok" se la scrittura va a buon fine. |
| 6) Controlla di aver ricevuto "Ok" da parte del server[2]. | |
| 7) Acquisisce l'oggetto e lo invia al server. | |
| | 8) Riceve l'oggetto, lo elabora e controlla che nella bacheca non sia già presente un altro messaggio con lo stesso oggetto; in caso affermativo, manda un "Ok". |
| 9) Controlla di aver ricevuto "Ok" da parte del server[3]. | |
| 10) Acquisisce il testo del messaggio e lo invia al server. | |
| | 11) Riceve il testo, lo elabora e inserisce l'intero messaggio nella bacheca.<br>Manda un "Ok" se la scrittura va a buon fine e ritorna. |
| 12) Controlla di aver ricevuto "Ok" | |

| | |
|---|---|
| da parte del server[4] e ritorna. | |
| **Note/Gestione degli errori**<br>[1]: se riceve una stringa diversa da "Ok" (ci sono stati problemi durante l'invio/ricezione), viene chiamata `ERROR_HELPER`.<br>[2]: nel caso in cui il server non abbia scritto niente nel file, si torna al punto 4.<br>[3]: se riceve la stringa "Kk" dal server, vuol dire che esiste già un messaggio con lo stesso oggetto e si torna al punto 7, chiedendo all'utente di inserirne uno diverso; se riceve un "Ko", il server non ha scritto niente sul file e si torna comunque al punto 7.<br>[4]: nel caso in cui riceva una stringa diversa da "Ok", l'utente è invitato a ripetere l'operazione da capo. | |

### 3.4 Cancellazione di un messaggio

| Client | Server |
|---|---|
| 1) Invia la stringa comando "3" al server per avvertirlo che l'utente vuole eliminare un messaggio. | |
| | 2) Risponde con la stringa "OK". |
| 3) Controlla di aver ricevuto "Ok" da parte del server[1]. | |
| 4) Invia il proprio username al server. | |
| | 5) Riceve lo username, controlla che sia presente nel file delle credenziali (controllo errore di invio/ricezione).<br>Manda un "Ok" se è presente.<br>Elabora lo username per la successiva autorizzazione a cancellare il messaggio. |
| 6) Controlla di aver ricevuto "Ok" da parte del server[2]. | |
| 7) Acquisisce l'oggetto e lo invia al server. | |
| | 8) Riceve l'oggetto e lo elabora.<br>Controlla:<br>a) che esista un messaggio con quello specifico oggetto;<br>b) se ne segna la posizione che questo occupa all'interno della bacheca e controlla se l'utente possieda l'autorizzazione per |

| | |
|---|---|
| | poterlo eliminare.<br>Lo elimina dalla bacheca, manda un "Ok" e ritorna. |
| 9) Controlla di aver ricevuto "Ok" da parte del server[3] e ritorna. | |
| **Note/Gestione degli errori**<br>[1]: se riceve una stringa diversa da "Ok" (ci sono stati problemi durante l'invio/ricezione), viene chiamata `ERROR_HELPER`.<br>[2]: nel caso in cui il server non trovi il mittente nel file, si ritenta l'invio tornando al punto 4.<br>[3]: se riceve la stringa "Kk" dal server, il messaggio esiste ma l'utente non possiede l'autorizzazione per eliminarlo, non essendone il mittente, ed entrambi ritornano; se riceve "Ne" vuol dire che non esiste alcun messaggio con l'oggetto ricevuto; se riceve altro (ci sono stati problemi durante l'invio/ricezione), viene chiamata `ERROR_HELPER`. | |

### 3.5 Chiusura della connessione

| Client | Server |
|---|---|
| 1) Invia la stringa comando "4" al server per avvertirlo che l'utente vuole terminare la connessione. | |
| | 2) Risponde con la stringa "Ok".<br>Termina il thread e ritorna. |
| 3) Controlla di aver ricevuto "Ok" da parte del server[1].<br>Chiude la socket e termina con successo. | |
| **Note/Gestione degli errori**<br>[1]: se riceve una stringa diversa da "Ok" (ci sono stati problemi durante l'invio/ricezione), viene chiamata `ERROR_HELPER`. | |

## 4. Gestione dei segnali e degli errori delle funzioni di sistema

Nel *server*, per i **segnali** `SIGHUP`, `SIGINT`, `SIGQUIT`, `SIGILL`, `SIGSEGV` e `SIGTERM` viene lanciata la funzione `server_close_handler`, che si occupa della terminazione sicura del server, operando la chiusura di descrittori, semafori e della socket.
Nel caso del segnale `SIGPIPE`, il server, invece, non termina l'esecuzione.

Nel *client*, per i segnali `SIGHUP`, `SIGINT`, `SIGQUIT`, `SIGILL`, `SIGSEGV` e `SIGTERM` viene chiamata la funzione `client_close`, che si occupa di chiudere il descrittore della socket terminando il client in maniera corretta.

Il segnale `SIGPIPE`, invece, viene ignorato.

Nel caso in cui falliscano le **funzioni di sistema** (`send`, `receive`, `write`, …), viene sempre chiamata `ERROR_HELPER`.

## 5. Istruzioni per l'avviamento

Per avviare il **server**, aprire una finestra del terminale nella cartella del progetto (o spostarvisi dentro), digitare `make` e premere il tasto Invio.

Per avviare il **client**, aprire una finestra del terminale nella cartella del progetto (o spostarvisi dentro), digitare `./client [IP]` (con il parametro `IP` opzionale) e premere il tasto Invio.