Bacheca Elettronica Remota

Il servizio permette ad un utente autentificato di inserire all'interno di una bacheca elettronica un messaggio il quale può essere letto da ogni altro utente interessato a consultare la bacheca stessa.
Il server accetta e sequenzia in concorrenza tutte le richieste provenienti da client su diverse macchine.
Un utente che si connette tramite l'apposito client può:

- Leggere tutti i messaggi presenti sulla bacheca elettronica
- Inserire un nuovo messaggio nella bacheca elettronica
- Rimuovere un messaggio dalla bacheca elettronica se inserito dallo stesso utente tramite verifica con password

Ogni messaggio (Post) che viene inserito sulla bacheca contiene un oggetto, un testo, una password ed il mittente.

Sono inclusi i seguenti file:

- server.c
- client.c
- lib.h
- thread.c
- thread.h
- showcase_util.c
- showcase_util.h
- socket_util.c
- socket_util.h
- Makefile

Server.c

Il server include la libreria lib.h che a sua volta include gli altri file richiesti.
Appena inizializzato il server si prepara a gestire i segnali in arrivo tramite la funzione *signal* che una volta ricevuto il segnale richiamerà la funzione *exit_server.* Quest'ultima viene chiamata solo in caso di errori gravi nel server e garantisce una chiusura sicura del server: chiude la socket principale del server, libera la memoria inizializzata per la bacheca, chiude il semaforo binario e chiude il server.

A cura di Giacomo Ceribelli

Successivamente viene allocata la memoria dedicata al semaforo di tipo named e in caso di errore tenta di riaprilo, se l'errore persiste il server si chiude.

Viene creata la socket per accettare nuove connessioni e vengono impostati i parametri di accettazione della struttura *sockaddr_in.*
Viene poi associata la struttura al descrittore della socket e assegnata all'indirizzo del localhost. In seguito si predispone la socket all'ascolto tramite la funzione *listen.*

Inizia ora il loop infinito in attesa di nuove connessioni sulla socket principale. Una volta controllati gli errori viene inizializzata la struttura per un thread al quale viene passate le informazioni riguardanti il client da gestire.
Il thread viene poi fatto partire ed il server si mette di novo in attesa.

In caso di uscita dal loop il server termia con successo.

Thread.c

Per ogni nuovo client che si connette viene creato un nuovo thread e viene cheamata la funzione *client_handler*.

Vengono salvati i parametri presi in input e inizializzate le variabili dedicate alla gestione della socket e i vari tipi di comandi possibili del client.
Tramite una struttura di tipo *timeval* dove viene settato un timeout di 10 minuti sulla socket tramite le funzioni *setsockopt.*
In caso di errore viene chiamata la funzione *exit_thread* che chiude la socket dedicata a quel client e libera la memoria riguardante la struttura del client.

Per prima cosa sulla socket viene mandato un messaggio di richiesta di username al client, l'username viene poi salvato nella variabile username una volta ricevuta.

Successivamente viene inviata la lista di tutti i messaggi presenti sul server tramite la funzione *send_showcase.*
A questo punto inizia il ciclo e ci si mette in attesa di comando dal client.
Rispettivamente i comandi ricevuti chiamato le funzioni:

- SHOW -> send_showcase
- NEW -> recv_post
- DELETE -> recv_delete
- QUIT -> interrompe il ciclo

Altrimenti in caso di comando sconosciuto si manda l'errore al client
Se il comando è quello di uscite il ciclo viene interrotto e viene chiamata *exit_thread.*

Socket_util.c

Il file socket_util contiene tutte le funzioni dedicate alla gestione della socket sia da parte del server che da parte del client.

Sock_error è una funzione booleana che controlla se i byte ricevuti nella socket sono <0 quindi c'è stato un errore sulla socket.

Send_showcase si preoccupa di inviare la bacheca sulla socket.
Per problemi di concorrenza si utilizza il semaforo per accedere alla bacheca ed inviare un singolo post alla volta e ci si mette in attesa di un ACK da parte del client.
Una volta terminato l'invio si invia il messaggio "finished" che implica la fine dell'invio.

Recv_post ha l'obiettivo di ricevete un nuovo post ed una volta ottenute le informazioni il post viene inserito all'interno della bacheca.
Vengono inizializzate le variabili di testo, oggetto, password…
Viene eseguita ora la procedura di ricezione, salvataggio ed invio dell'ACK per ogni informazione.
A questo punto viene inserito il post nella bacheca tramite la funzione *insert_post* . Viene quindi inviato il messaggio di esito d'inserimento al client.

Recv_delete serve ad eliminare un post dalla bacheca elettronica.
Vengono inizializzate le variabili di password, id e altri buffer, si riceva l'id e la password i quali vengono salvati negli appositi buffer (viene inviato l'ACK).
Si chiama *delete_post* per eliminare il post.
Successivamente viene inviato l'esito (se negativo con tipo di errore).

Recv_showcase è chiamata dal client e si occupa di ricevere e stampare la bacheca.
Viene avviato un ciclo nel quale in ordine si riceve un post che se diverso da "finished" viene stampato, altrimenti il ciclo viene interrotto.
Ogni post viene poi stampato e si invia l'ACK per confermare la ricezione.

Send_post è chiamata dal client e si occupa di inviare un post al server.
Il primo ciclo si occupa di catturare l'oggetto del post.

In caso l'oggetto sia troppo lungo o troppo corto si continua con il ciclo, altrimenti se accettabile viene interrotto e viene inviato. A questo punto il client si mette in attesa dell'ACK da parte del server. La stessa procedure è eseguita per testo e password che vengono richiesti successivamente.

Send_delete è chiamata dal client e si occupa di inviare la richiesta di cancellazione del post.
Nel primo ciclo viene richiesto l'ID del post da eliminare, viene effettuato il controllo se il testo inserito corrisponde a caratteri numerici o se non è stato inserito nulla, altrimenti il ciclo viene interrotto.
Successivamente si invia l'ID e ci si mette in attesa dell'ACK da parte del server. La stessa procedura viene eseguita sulla cattura della password.

Showcase_util.c

Il file showcase_util.c si occupa di lavorare esclusivamente sulla bacheca, contiene infatti solo funzioni che operano direttamente sulla bacheca elttronica.

Empty_showcase è una funzione booleana che controlla se la bacheca è vuota.

Insert_post riceve in input username, oggetto, testo e password e ha l'obiettivo di aggiugnere un nuovo post in coda alla bacheca.
Viene creato un nuovo elemento di tipo *showcase,* gli input vengono inseriti all'interno del nuovo elemento insieme alla data attuale.
Se la bacheca è vuota è il primo elemento pertanto la bacheca viene inizializzata al nuovo elemento, altrimenti si scorre la bacheca fino all'ultimo elemento, e il nuovo elemento viene appeso alla fine con numero di ID successivo all'ultimo.
In entrambi i casi viene chiamata prima *semaphore_wait* e dopo *semaphore_post* per decrementare e aumentare rispettivamente il valore del semaforo per problemi di concorrenza.
Viene ritornato 1 se il post è stato inserito con successo, 0 altrimenti.

Delete_post prende in input un ID, una password ed un username; ritorna un puntatore di char e ha l'obiettivo di rimuovere un post dalla bacheca.
Si controlla che la bacheca non sia vuota altrimenti viene ritornato il messaggio di bacheca vuota.
Con un ciclo si procede alla ricerca dell'elemento con il rispettivo id, altrimenti viene ritornato il messaggio di post inesistente.

Una volta trovato l'id viene eseguito il controllo sull'username, viene ritornato il messaggio di username errato se il controllo non è superato.
L'ultimo controllo viene eseguito sulla password altrimenti viene ritornato il messaggio di password errata.
L'elemento viene quindi eliminato e la memoria liberata, viene ritornato il messaggio di cancellazione avvenuta con successo.
In tutti i casi viene chiamata prima *semaphore_wait* e dopo *semaphore_post* per decrementare e aumentare rispettivamente il valore del semaforo per problemi di concorrenza.

Showcase_util.h

È il file header della rispettiva bacheca, oltre a contenere le funzioni dichiarate nel file .c contiene le variabili globali della bacheca e del semaforo.
Inoltre viene anche dichiarato il tipo della struttura della bacheca: un struttura collegata formata da post.
La struttura post consiste in

- Id: un intero che indica il codice univoco del post
- Autore: un array di 15 caratteri contenente il nome dell'autore.
- Password: un array di 15 caratteri contente la password per l'eliminazione.
- Oggetto: un array di 50 caratteri contente l'oggetto del post.
- Testo: un array di 1900 caratteri contente il testo del post.
- Data: data e ora di inserimento del post.

La dimensione massima dei campi è stata pensata al fine di evitare il superamento dei 4096 byte durante l'invio tramite socket sommati ai rispettivi caratteri che vengono aggiunti durante l'invio di un singolo post su socket.

Client.c

È il file che contiene il codice relativo un client che si connette al server.
Vengono chiamate inizialmente le *signal* per catturare eventuali segnali e garantire un'uscita sicura dal programma tramite la funzione *exit_client* la quale chiude la socket.
Vengono inizializzati i vari tipi di comando accettabili e le strutture per la socket. Viene creata la socket e inseriti i valori all'interno della struttura *sockeddr_in* per il collegamento con il server.
Tramite la funzione *connect* si avvia la connessione sulla socket.
Ci si mette in attesa del primo messaggio da parte del server e viene stampato.

Un ciclo tenta di catturare l'username da inviare al server. Vengono effettuati i controlli su lunghezza minima e massima, successivamente viene inviato al server su socket.
Si riceve un secondo messaggio di benvenuto dal server che precede la chiamata a *recv_showcase* la quale riceve la bacheca da stampare.
Inizia il ciclo che stampa i vari tipi di comandi che l'utente può inserire. Viene catturato un comando che viene inviato al server. Se il comando è

- SHOW viene chiamata nuovamente *recv_showcase*.
- NEW viene chiamata la funzione *send_post*.
- DELETE viene chiamata la funzione *send_delete*.
- QUIT il ciclo viene interrotto.

Successivamente si attende risposta dal server con il relativo esito del comando inviato.
Se il ciclo viene interrotto viene chiamata la funzione *exit_client*.

Lib.h

È il file header che contiene l'importazione alle altre librerie richieste dal programma. Contiene inoltre la struttura handler_args_t che presenta le informazioni utili ad un thread per gestire il suo client.
È presente inoltre una funzione ERROR_HELPER che viene chiamata in rari casi, essa in caso di errore stampa il messaggio e chiude il rispettivo programma sia server che client.
Sono presenti anche le variabili globali utili alla gestione della connessione sia del server che del client.

Per eseguire il programma basta posizionarsi all'interno della cartella contente i file nel terminale, digitare "*make run*" per avviare il server in una finestra di terminale e "*./client*" per avviare il client da un'altra finestra di terminale