Salvatore Foderaro - Matricola n°0214381

Corso di laurea “Ingegneria Informatica” - AA 2018/2019

Bacheca Elettronica

Panoramica

Questa tesina è stata realizzata da Salvatore Foderaro (Matricola n° 0214381) per il corso di **Sistemi Operativi,** tenuto dal professore **Francesco Quaglia** nell’AA 2017/2018. Lo sviluppo del progetto ha permesso di applicare concetti inerenti la programmazione C in ambiente Linux, affrontati a lezione.

Specifica

Realizzazione di un servizio bacheca elettronica che permetta ad ogni utente autorizzato di inviare messaggi che possono essere letti da ogni altro utente interessato a consultare la bacheca stessa. Il servizio di accesso alla bacheca elettronica deve essere offerto da un server che accetta e processa sequenzialmente o in concorrenza le richieste dei client.

Obiettivi

1. **Lettura dei messaggi:** l’utente che accede alla bacheca elettronica deve avere la possibilità di leggere tutti i messaggi presenti.
2. **Inserimento di un nuovo messaggio:** l’utente può inviare un nuovo messaggio contenente i campi *mittente, oggetto e testo***.**
3. **Rimozione di un messaggio**: l’utente ha la possibilità di eliminare un messaggio dalla bacheca, se il messaggio in oggetto è stato precedentemente da lui stesso inviato.

Scelte di progetto

1. Gestione multithreading / Gestione concorrenza

Il server opera in concorrenza, affidando ad ogni singolo thread la gestione della singola connessione in entrata. Per la gestione della concorrenza vengono utilizzati due semafori (*fileMessagesAccess* e *fileUsersAccess*) per la scrittura su file, oltre ad un singolo semaforo (*counter*) per tenere traccia del numero di utenti collegati.

2. Gestione dei segnali

Per quanto riguarda il server, i segnali SIGACTION, SIGQUIT e SIGTERM sono bloccati per tutti i thread, escluso *sig_thread* che, in attesa dell'arrivo di uno dei tre segnali, ha il ruolo di gestore. Per evitare problemi di consistenza dovuti all'interruzione delle scritture su file, ricevuto uno dei tre segnali, *sig_thread* attende il termine delle scritture su file e successivamente termina l'applicazione.

Nel caso di ricezione del segnale SIGKILL durante una scrittura, ad ogni avvio del server viene incaricato un thread (*recupero_consistenza_file*) per la rimozione dei byte successivi alla scrittura incompleta dei file **user.dat** e **messages.dat**. La gestione dell'interruzione della connessione (sia lato client che lato server) viene gestita ignorando il segnale SIGPIPE e controllando il numero di byte ricevuti/inviati da due le parti.

3. Gestione dei dati

Per garantire la persistenza dei dati vengono utilizzati due file distinti, **messages.dat** e **user.dat.** All'interno dei files i dati vengono memorizzati sotto forma di struttura, utilizzando la funzione *fwrite*, per essere successivamente consultati/resi fruibili al sistema, sfruttando la funzione *fopen*.

4. Gestione sessione utente

La struttura ***utente_loggato*** (del tipo *comunicazione*), viene mantenuta come variabile locale da ogni thread. Una volta effettuato il login verranno memorizzate al suo interno l'ID ed il nome dell'utente, riutilizzate dal server per le operazioni che, per essere eseguite, necessitano dell'accesso al sistema.

Per garantire la consistenza delle informazioni relative alla sessione (ultimo ID utente/ultimo ID messaggio inserito), all'avvio del server viene incaricato un thread

(*recupero_consistenza_sessione*) del recupero di tali informazioni, leggendo i file **user.dat/messages.dat** ed inserendoli nella struttura **session** (*consistenzaSessione*) dichiarata come variabile globale.

5. Comunicazione Client-Server

La comunicazione tra client e server, di tipo *SOCK_STREAM*, avviene tramite dominio *AF_INET*. Per la richiesta di operazioni da parte del client vengono utilizzate le strutture **comunicazione,** i cui campi sono esposti nella tabella 1, e **messaggi**; le due strutture vengono codificate in stringa (funzione *encodeCommunication*) da parte del mittente, e decodificate (funzione *decodeCommunication*) dal destinatario utilizzando la funzione *strtok*. Prima della trasmissione, tutti i numeri vengono convertiti in *network byte order.*

*Tabella 1: Valore dei campi della struttura **comunicazione** per ogni operazione permessa al client*

Operazione	Codice Operazione	Argomento 1	Argomento 2
Effettua accesso	0	Nome utente	Password
Leggi tutti i messaggi	1	NULL	NULL
Inserisci messaggio	2	Oggetto messaggio	Testo messaggio
Rimuovi messaggio	3	ID messaggio	NULL
Effettua registrazione	4	Nome utente	Password
Disconnessione	5	NULL	NULL

Il server risponde alle richieste del client con uno dei valori di ritorno presenti nella lista completa, disponibile come commento iniziale al file **Server.c**.

Documentazione

Come compilare

Per la compilazione dei sorgenti è disponibile il Makefile con un parametro per client ed uno per il server.

```
make client && make server
```

Comandi disponibili

```
./Server numero_porta // Avvia il server utilizzando la porta specificata
./Server --seeuser // Visualizza la lista degli utenti registrati nel sistema
./Client 127.0.0.1 numero_porta // Effettua la connessione con il server avviato in locale
./Client indirizzo_server numero_porta // Effettua la connessione con il Server
```