Sistemi di Calcolo - Modulo 2 (A.A. 2014-2015)

Simulazione prova al calcolatore - 21 maggio 2015

Tempo a disposizione: 1h 30'.

Nota bene: assicurarsi di compilare il file **studente.txt** e che il codice prodotto non contenga errori di compilazione, pena una valutazione negativa dell'elaborato.

Realizzazione di una chat room

L'obiettivo di questa prova è completare il codice della parte server di una chat room.

Una chat room è un'applicazione che permette ad ogni suo utente di inviare messaggi che siano visibili in tempo reale a tutti gli altri partecipanti. Un utente si connette al server tramite un client specificando il nickname che vorrebbe utilizzare, e nel caso questo sia disponibile il server invia un messaggio di benvenuto con le informazioni essenziali su come inviare comandi al server e messaggi agli altri utenti.

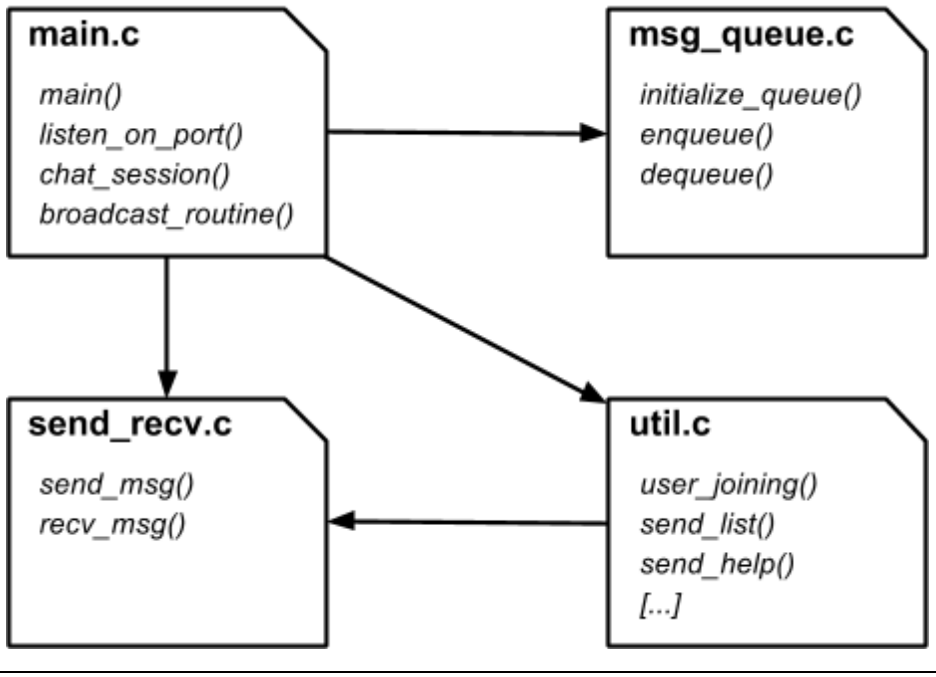
Descrizione del codice del server

Il codice lato server implementa una serie di operazioni di base, tra le quali:

- gestire richieste di connessione dei client e mantenere le informazioni associate a ciascun utente (nickname, descrittore della socket, indirizzo IP e porta, etc.);
- inoltrare agli altri utenti ogni nuovo messaggio inviato da un utente;
- notificare agli utenti la connessione o disconnessione di un utente;
- rispondere a richieste per il server inviate da un utente (mostrare la lista degli utenti connessi, inviare statistiche sulla connessione, fornire un messaggio di help).

Per ottimizzare la stesura e la leggibilità del codice, il server è organizzato in modo modulare, con due file di header `.h` e quattro file contenenti i sorgenti `.c` veri e propri:

common.h	Definizione delle strutture dati e delle macro (incluse quelle per la gestione degli errori) in uso nei sorgenti `.c`
methods.h	Contiene i prototipi dei metodi definiti in ciascun modulo
main.c	Avvio del server e gestione delle connessioni in ingresso
msg_queue.c	Metodi per la gestione thread-safe di una coda di messaggi
send_recv.c	Metodi per l'invio e la ricezione dei dati su socket
util.c	Miscellanea (registrazione di un nuovo utente, gestione dei comandi inviati al server, metodi ausiliari per le operazioni interne al server)
Makefile	Compilazione automatizzata del codice



Architettura del codice sorgente. Una freccia indica che il modulo di origine utilizza uno o più metodi definiti nel modulo di destinazione.

Obiettivi
Vi viene richiesto di completare porzioni di codice - contrassegnate anche da blocchi di commenti nei rispettivi file - nei sorgenti C per implementare le seguenti operazioni:

(1) All'interno del metodo `main()` nel modulo `main.c`, creare un nuovo thread che esegua la routine `broadcast_routine()`, che non necessita di alcun argomento da passare. Nel prosieguo del programma non sono previste operazioni di join su tale thread.

(2) All'interno del metodo `listen_on_port()` nel modulo `main.c`, creare un nuovo thread che esegua la routine `chat_session()`. Per questo scopo sarà necessario allocare dinamicamente un buffer per un oggetto `session_thread_args_t`, i cui campi `socket` ed `address` andranno inizializzati con i valori corrispondenti definiti nel corpo del `while (1)`, e passare il puntatore al buffer come argomento durante la creazione del thread. Nel prosieguo del programma non sono previste operazioni di join su tale thread.

(3) Il modulo `msg_queue.c` implementa una struttura dati coda tramite buffer circolare.

Il metodo `initialize_queue()` inizializza la struttura dati; il metodo `enqueue()` inserisce un messaggio nella coda, mentre il metodo `dequeue()` estrae il primo messaggio non ancora letto secondo una politica FIFO (First In First Out).

Modificare i tre metodi inserendo operazioni su semafori che impediscano *race conditions*. In particolare, più thread possono invocare contemporaneamente il metodo `enqueue()`, mentre è soltanto uno il thread - quello creato al punto (1) - a poter invocare il metodo `dequeue()`. Si consiglia di dichiarare i semafori come variabili globali all'inizio del modulo, e non è richiesto di inserire alcun codice per la distruzione di tali semafori.

(4) Il modulo `send_recv.c` contiene due primitive per la comunicazione su socket.

Si completi il metodo `send_msg()` affinché il contenuto del buffer `msg_to_send` (generato internamente a partire dal contenuto dell'argomento `msg`) venga scritto sul descrittore argomento `socket`. Una implementazione che si assicuri che tutti i byte siano stati inviati verrà premiata in sede di valutazione.

Si completi poi il metodo `recv_msg()` affinché scriva nel buffer `buf` un messaggio di lunghezza fino a `buf_len` bytes leggendolo dal descrittore `socket`. La fine di ogni messaggio è contrassegnata dal carattere di ritorno a capo `'\n'` e pertanto si proceda leggendo un unico byte per volta dal descrittore e copiandolo nel buffer, finché il carattere di ritorno a capo non viene incontrato. Nel caso in cui la socket venga chiusa dall'endpoint il metodo deve restituire immediatamente `-1`, altrimenti al termine della lettura il numero di byte letti (escluso il carattere `'\n'`) memorizzato nella variabile locale `bytes_read`.

Testing
La compilazione è incrementale (ossia avviene un modulo per volta) ed automatizzata tramite `Makefile`, pertanto in assenza di errori di compilazione un binario `server` verrà generato quando si esegue `make`.

Si può lanciare ad esempio il server sulla porta 1025 con:

```
./server 1025
```

Viene fornito un eseguibile `client` precompilato che può essere utilizzato per testare rapidamente il corretto funzionamento del server. Per connettersi con nickname `pippo` al server in esecuzione sulla macchina locale si può quindi scrivere in un nuovo terminale:

```
./client 127.0.0.1 1025 pippo
```

In un terzo terminale si può aprire una nuova connessione con nickname `pluto`:

```
./client 127.0.0.1 1025 pluto
```

A questo punto sul terminale in uso al client `pippo` comparirà un messaggio di notifica che avviserà dell'avvenuta connessione dell'utente `pluto` alla chat room. Ogni utente può mandare un messaggio semplicemente scrivendolo su terminale e premendo Invio.
Per ottenere la lista degli utenti connessi si scriva il comando `#list` e si prema Invio, mentre per uscire dalla chat room e quindi dal client si scriva il comando `#quit` seguito da Invio. Quando uno dei client connessi esce dalla chat room, gli altri utenti ancora connessi riceveranno un messaggio di notifica dal server.

Nota: se un client viene chiuso con CTRL-C, il server andrà in crash. Questo perché quando si tenta di scrivere su una connessione chiusa, il sistema genera un segnale `SIGPIPE`. Per ovviare a ciò il server avrebbe dovuto catturare il segnale e quindi gestire errori di tipo `EPIPE` per `send()`: tali aspetti vanno oltre gli obiettivi di questo corso.

Altro

- in caso di cancellazione accidentale di uno o più file, nella cartella `backup/` è presente una copia di sicurezza della traccia di esame
- il file `dispensa.pdf` contiene una copia della dispensa *Primitive C per UNIX Sytem Programming* preparata dai tutor di questo corso