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

Prova al calcolatore di fine corso - 28 maggio 2015

Tempo a disposizione: 1h 30'.

Attenzione: 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 uno YELL Server

L'obiettivo di questa prova è completare il codice di uno YELL Server. Uno YELL Server riceve messaggi dai suoi client e risponde con messaggi ottenuti modificando quelli ricevuti come segue:

- tutte le lettere vengono messe in maiuscolo
- vengono aggiunti tanti punti esclamativi quanti sono i caratteri del messaggio

Lo YELL Server si mette in ascolto di connessioni TCP su una determinata porta. I client si connettono specificando la porta del server. Un client può terminare la connessione inviando il messaggio 'BYE'. Il server impone un limite al numero massimo di client contemporaneamente connessi. Inoltre, il server memorizza su un file di log tutti i messaggi ricevuti dai client.

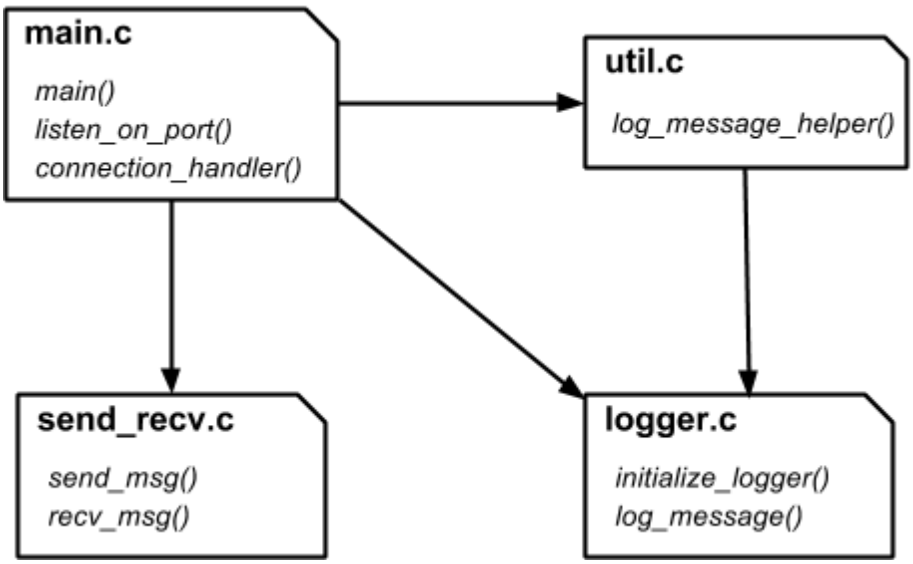
Descrizione del codice del server

Il codice lato server implementa queste operazioni di base:

- gestire richieste di connessione dei client;
- imporre un limite al numero massimo di client serviti contemporaneamente;
- rispondere ai messaggi inviati dai client;
- loggare su file i messaggi ricevuti dai client;
- chiudere la connessione con un client se questi invia il messaggio 'BYE'.

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
send_recv.c	Metodi per l'invio e la ricezione dei dati su socket
logger.c	Metodi per loggare su file e per inizializzare le necessarie strutture dati di controllo
util.c	Miscellanea (metodo helper per la formattazione dei messaggi di log)
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`, gestire l'apertura del semaforo da usare per il controllo del numero massimo di accessi concorrenti.

(2) All'interno del metodo `listen_on_port()` nel modulo `main.c`, creare un processo figlio per ogni connessione client accettata. Contestualmente, implementare il controllo del numero di accessi concorrenti. Porre particolare attenzione al rilascio delle risorse non necessarie, sia nel processo padre che nel figlio.

(3) Il modulo `logger.c` include le funzioni da usare per la gestione del logging. Il metodo `initialize_logger()` si occupa dell'apertura del file di log e dell'inizializzazione del semaforo per controllare l'accesso in scrittura al file stesso. Il metodo `log_message()` scrive sul file di log un nuovo messaggio in modo tale da evitare race conditions. Completare l'implementazione di queste due funzioni.

(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` venga scritto sul descrittore argomento `socket`. Una implementazione che assicuri che tutti i `msg_len` 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` byte leggendolo dal descrittore `socket`. Nel caso in cui la

socket venga chiusa dall'endpoint, il metodo deve restituire immediatamente `-1`, altrimenti il numero di byte realmente letti.

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 al server in esecuzione sulla macchina locale si può quindi scrivere in un nuovo terminale:

```
./client 1025
```

Per lanciare più client al fine di testare le routine di controllo sul numero di connessioni concorrenti, è possibile lanciare lo stesso comando su più terminali.

Ogni client può mandare un messaggio semplicemente scrivendolo su terminale e premendo Invio. Per terminare una connessione client, inviare il messaggio 'BYE'.

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

---

Raccomandazioni
Seguono alcune considerazioni sugli errori riscontrati più di frequente tra gli elaborati dei partecipanti, nonché delle raccomandazioni per un buon esito delle prove al calcolatore:

- dato un `char* buf`, `sizeof(buf)` non restituirà il numero di byte presenti nel buffer, bensì il numero di byte occupati da un puntatore (4 su IA32, 8 su x86_64); se `buf` contiene una stringa NULL-terminated è possibile utilizzare `strlen(buf)`

- quando si deve invocare una funzione per la programmazione di sistema:
  - fare attenzione se i singoli parametri siano o meno dei puntatori
  - verificare dalla documentazione se può essere soggetta ad interruzioni, e in tal caso gestirle opportunamente
  - verificare il comportamento della funzione in caso di errori: inserire codice idoneo a trattare errori gestibili, utilizzare la macro opportuna per gli altri
  - per funzioni della libreria pthread, si vedano il capitolo 2 e l'appendice A.2 della dispensa di UNIX system programming

- quando viene acquisita una risorsa (socket aperta, area di memoria allocata, etc.), questa deve poi essere rilasciata in maniera opportuna

- utilizzare puntatori non inizializzati porta ragionevolmente un Segmentation fault!

- i cast di puntatori da e verso `void*` sono impliciti in C, renderli espliciti o meno è una scelta di natura puramente stilistica nella programmazione
- fare molta attenzione nell'aggiornamento dell'indice (o del puntatore) con cui si opera su un buffer: disallineamenti ±1 sono tra le cause più comuni di errori
- quando si vuole passare un puntatore ad una funzione, aggiungere al nome della funzione una coppia di parentesi tonde equivale invece ad invocarla!
- **i commenti nel codice contengono molte informazioni utili per lo svolgimento della prova, si consiglia quindi di tenerli in debita considerazione**

Regole Esame

- Domande ammesse
  Le domande possono riguardare solo la specifica dell'esame e la struttura di alto livello del codice, nessuna domanda può riguardare singole istruzioni.
- Oggetti vietati
  I seguenti oggetti non devono essere presenti sulla scrivania, né tantomeno usati: smartphone, telefonini, tablet, portatili, dispositivi di archiviazione USB, copie cartacee della dispensa, astucci e qualsiasi forma di libri ed appunti. **Chi verrà sorpreso ad usare uno di questi oggetti verrà automaticamente espulso dall'esame.**
- Azioni vietate
  È assolutamente vietato comunicare (in forma scritta, verbale o elettronica) con gli altri studenti. **Chi verrà sorpreso a comunicare con gli altri studenti per la prima volta verrà richiamato, la seconda volta verrà invece automaticamente espulso dall'esame**.