Reti 1

Relazione del Progetto di Laboratorio - Anno 2019/2020

Alessandro Borsoi 20014853

Compilazione e avvio

La compilazione avviene tramite `make`. Per compilare è possibile usare il comando `make all` mentre `make clean` elimina tutti gli artefatti di compilazione. L'applicazione server e l'applicazione client risiedono nella directory `apps` per cui, dopo aver compilato, è sufficiente eseguire dalla radice del progetto i due programmi tramite `./apps/server <numero_porta>` e `./apps/client <indirizzo_server> <numero_porta>` come indicato nelle specifiche.

Sono stati creati anche degli unit test, presenti nella directory `test`, che è possibile avviare con `./test/test_<nome_componente>`. Il comando `make all` compila in automatico anche loro.

Il progetto è stato sviluppato quasi interamente su un MacBook Pro con `gcc` e i flag di compilazione `-Wall -Wextra -std=c11 -pedantic` ma la correttezza e assenza di warning è stata verificata anche su una macchina (Arch) Linux. Inoltre è stata verificata la correttezza di compilazione anche con il compilatore `clang`. In aggiunta è stato usato sulle due macchine anche `valgrind` con i flag `--tool=memcheck --leak-check=full` su tutti gli eseguibili del progetto (client, server e test) per verificare l'assenza di memory leaks o in generale problemi legati alla gestione della memoria.

Struttura del progetto

Il progetto si compone di diverse directory, lasciano nella radice solo il Makefile generale. Come già accennato, `apps` contiene le due applicazioni client e server vere e proprie. Queste dipendono da liberie il cui header pubblico è contenuto nella cartella `include` (importata in fase di compilazione). All'interno di `include` si distinguono i componenti utilizzati dal client e dal server con un unico header comune `protocol.h` che definisce la dimensione dei messaggi del protocollo e degli stati possibili. Il codice sorgente di questi header è contenuto in `src`, compilato come libreria in `bin` e linkato staticamente. È stata fatta questa scelta per esporre una interfaccia facilmente testabile per dei componenti di utilità per i due programmi principali. Questi componenti sono:

- `store` usato dal server, astrae un generico repository per i numeri in ingresso ed espone il calcolo di media e varianza. Internamente è implementato come uno stack;
- `protocol` implemeta di fatto le logiche di ricevimento e risposta lato server, con lo store come dipendenza. Questo facilita la testabilità riducendo la logica ad una funzione pura;
- `splitter` usato dal client, implementa la logica di partizionamento in più messaggi di una sequenza numerica (anche molto lunga) inserita dall'utente tramite file di testo.

Nella directory `test` ci sono gli unit test dei tre componenti appena menzionati. Infine in `data` ci sono dei file di testo usati sia negli unit test che nei test manuali delle due applicazioni.

Client

Server