Relazione Progetto
Sistemi Operativi e Laboratorio

Samuele Calugi
Corso A
579086

Luglio 2021

Contents

1 Introduzione

Come previsto dalle indicazioni dell'esame, il progetto di seguito è stato sviluppato nella versione ridotta e semplificata poiché la parte facoltativa non è prevista nell'appello di Luglio 2021. Il codice sorgente e i vari file necessari per il corretto funzionamento del programma sono stati caricati in una repository pubblica su Github. È possibile accedere alla repository del progetto attraverso il seguente link: `https://github.com/Walrus98/Progetto-SOL`, così da poter visualizzare il codice sorgente e tutti i commit effettuati.
Le librerie utilizzate nel progetto sono le seguenti:

- **icl_hash.h**, libreria vista a lezione utilizzata per la creazione di mappe.
- **list_utils.h**, libreria realizzata dal sottoscritto utilizzata per la creazione di liste monodirezionali.

1.1 Makefile

L'intero progetto può essere compilato ed eseguito attraverso le seguenti regole:

- **all**: viene chiamato per default dal comando **make**, compila il client e il server attraverso le regole **build-client** e **build-server**.
- **build-client**: compila tutti i file del client e genera l'eseguile.
- **build-server**: compila tutti i file del server e genera l'eseguile.
- **build-server-test1**: compila tutti i file del server con i config richiesti da test1 e genera l'eseguile.
- **build-server-test2**: compila tutti i file del server con i config richiesti da test2 e genera l'eseguile.
- **client**: esegue un clear della console e successivamente avvia il client con valgrind.
- **server**: esegue un clear della console e successivamente avvia il server con valgrind.
- **server-test1**: esegue un clear della console e successivamente avvia il server con i config richiesti da test1 con valgrind.
- **server-test2**: esegue un clear della console e successivamente avvia il server con i config richiesti da test2 con valgrind.
- **clean**: rimuove tutti i file object generati dalle regole elencate precedentemente.

1.2 Test case

Nel progetto è presente una cartella di nome *"tests"* che al suo interno contiene i due file bash richiesti per testare il corretto funzionamento del programma. I due script una volta avviati compilano in maniera autonoma client e server tramite le regole del Makefile sopra citate. Successivamente, in base al tipo di file bash avviato, viene eseguito il server con il propio file config opportuno e un insieme di client che eseguono differenti richieste sul server.

2 Funzionamento del programma

2.1 API

Per potersi interfacciare con il server, il client include all'interno del suo codice un file header chiamato **client_network.h**. Quest'ultimo possiede l'elenco di tutti i metodi richiesti dal progetto dell'esame e permette quindi di per poter inviare e ricevere messaggi con il server attraverso l'utilizzo di un protocollo. Tutti i metodi del file header sono implementati nella classe **client_network.c**.

2.2 Protocollo

I messaggi trasmessi dal client al server sono formati da due parti:

- **Header**, contiene un id che identifica il tipo di pacchetto trasmesso e la dimensione del payload.

- **Payload**, il contenuto effettivo del pacchetto che si vuole trasmettere.

Di conseguenza, ogni pacchetto è formato da due buffer: il **BufferHeader** e il **BufferPayload**.

Il **BufferHeader** ha dimensione fissa di **8 Byte** di cui:

- i primi 4 byte, (quindi il primo intero) contengono l'id del pacchetto che si vuole trasmettere.
- i 4 byte successivi, (quindi il secondo intero) contengono la dimensione del payload.

Il **BufferPayload** ha una dimensione variabile e contiene tutti i dati che si vogliono trasmettere

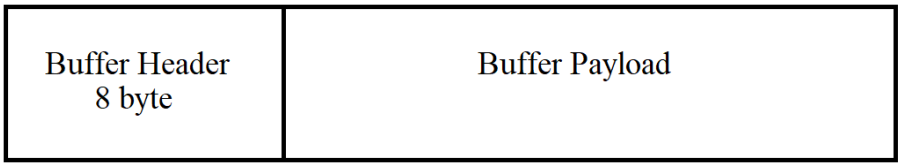

Figure 1: Stuttura del buffer di invio di messaggi al server

Poichè il client ogni volta che invia un messaggio al server si mette in attesa di ricevere un messaggio di risposta da quest'ultimo (quindi i pacchetti da parte del client vengono inviati in maniera sequenziale), il server non ha bisogno di inviare al client l'id del pacchetto perché il client conosce già il tipo di pacchetto che sta per ricevere. Di conseguenza, il server invia prima la dimensione del pacchetto, poi il contenuto del buffer che vuole inviare.
In alcuni casi, se il server non deve inviare nessun dato se non l'esito della richiesta che il client ha inviato, il server invia semplicemente un intero come messaggio di risposta: nel caso di successo, il server invia 0, in caso di errore un numero negativo.

2.3 Server

2.4 Client