myFileTransferApp.c

Introduzione

Il software si suddivide in un programma server e un programma client, di seguito li analizzeremo in quest'ordine soffermandoci sulle tre operazioni richieste all'applicazione di file transfer: **scrittura su server**, **lettura su server** e **lettura di directory**, con infine una breve digressione sulla **gestione degli errori**.

---

myServer.c

Il programma si suddivide in una funzione **main** e (escludendo funzioni d'appoggio minori) tre altre grandi funzioni: **write_local**, **read_local** e **ls_local**.

Nel **main** avviene la categorizzazione degli argomenti, l'eventuale generazione della directory passata come argomento e la creazione del socket TCP, il binding e il listening, con seguente accept e chiamata alla funzione richiesta dal client. È doveroso dire che nel main troviamo anche l'inizializzazione di un semaforo poiché, nel momento antecedente alla chiamata a funzione, avviene una fork, la quale consentirà la gestione di più richieste client in maniera contemporanea.

La funzione **write_local**, come è chiaro, *si occupa di ricevere e scrivere su disco un file dal client*. Per fare questo come prima cosa riceve il nome del file e si occupa di generarlo (anche con associata directory in caso non esistesse) e in seguito registra mano a mano i dati su disco, in caso di errori il file non verrà lasciato a metà ma verrà rimosso al fine di garantire integrità dei dati.

<<poiché si sta implementando un server che gestisce più client, sezioni di codice come questa sono molto delicate e, per questo, vengono implementate sezioni critiche attraverso i semafori, al fine di mantenere la coerenza delle informazioni>>

La funzione **read_local** *si occupa invece di ricercare e inviare il file richiesto dal client*, anche in questo caso la prima cosa da fare è registrare il nome del file e controllarne l'esistenza, una volta confermata al client, questo si prepara a ricevere e read_local inizia a inviare i buffer.
Anche in questo caso ci troviamo in una sezione critica e vengono sfruttati i semafori.

L'ultima funzione da analizzare è **ls_local**, leggermente diversa dalle altre due, *si occupa di inviare i contenuti* (e alcune informazioni su di essi) *di una specifica directory* richiesta dal client.
Come read_local la prima cosa che fa è confermare l'esistenza della directory al client, che si prepara a ricevere, e successivamente sfrutta il comando ls -la in locale attraverso execve e ne invia l'output al client.
Anche qui ci troviamo in una situazione in cui è importante mantenere l'integrità delle informazioni ed è dunque necessario l'ausilio dei semafori.

---

myClient.c

Il programma è quasi speculare alla sua versione server, una delle poche differenze è che, non essendo necessario per il client gestire più richieste e essendo leggermente più snello nella logica, non troviamo tre funzioni d'appoggio separate ma solamente tre sezioni all'interno del main volte a scrivere su server, leggere da server e ricevere la directory.
Per quanto riguarda l'inizializzazione c'è solamente la creazione del socket e il tentativo di connessione al server con conseguente richiesta (oltre che la categorizzazione degli argomenti in input).

Nella **sezione di scrittura** la prima cosa che avviene è l'invio della richiesta, sotto forma di codice ("w"), al server, il quale si prepara a ricevere il nome del file da scrivere e invia una conerma di creazione, in seguito alla quale è inviato il contenuto.

La **sezione di lettura** è paragonabile alla controparte write_local del server (come la funzione di scrittura lo è a read_local).
Anche qui, come nella sezione di scrittura, la prima cosa che avviene è l'invio del codice di lettura, che prepara il server all'ascolto, con conseguente invio del nome del file. A questo punto il client rimane in attesa della conferma di reperimento e, in seguito a riscontro positivo, inizia a ricevere e trascrivere.
Va aggiunto che nel caso in cui la directory di scrittura non esista già il programma si adopera a crearla.

L'ultima sezione è quella di **ricezione** dei contenuti della **directory**, questa agisce in maniera estremamente simile alla sezione di lettura, con unica differenza che invece di salvare il contenuto in un file, i dati ricevuti vengono direttamente stampati a schermo.

Gestione errori

Un programma di trasferimento file può incontrare vari problemi: i **problemi di rete** (binding, connection, etc.), di **spazio su disco**, di **omonimia**, di **scrittura/lettura concorrente** e **validità dei dati** inseriti.

Per quanto riguarda i **problemi di rete**, *le connessioni* (ovvero i vari send e recv) *vengono costantemente monitorate* e in caso di errore questo viene raccolto, printato e quella specifica funzione del server viene chiusa lasciando però il server in piedi.

Per lo **spazio su disco** il controllo avviene ovviamente solamente nei momenti di scrittura dei file (sia lato client che server), ad ogni nuovo buffer ricevuto infatti si controlla che lo spazio rimanente sia sufficiente alla scrittura su disco.

<<In caso di riscontro negativo è doveroso considerare i dati già inseriti, che vengono dunque rimossi al fine di conservare l'integrità delle informazioni e non avere residui di file incompleti>>

Sempre in materia di scrittura vanno considerate altre due situazioni: **omonimia** e **directory non esistente**.

L'omonimia viene risolta modificando il nome del nuovo file, al quale viene aggiunto un indice numerico relativo al numero di file presenti con quello stesso nome. Per la directory non esistente, sia lato server che lato client, è presente una funzione di creazione directory ricorsiva (dal lato server è una funzione d'appoggio definita), la quale controlla l'esistenza della directory e in caso negativo ne provvede alla creazione.

Ancora troviamo il problema di **scrittura/lettura concorrente**, essendo infatti il server un programma multi-processo è possibile due client diversi generino richieste conflittuali che potrebbero portare a file non validi.
Questo viene risolto attraverso l'utilizzo di un semaforo binario che evita due processi entrino entrambi in sezioni critiche nello stesso momento.

Per ultimo la **validità dei dati inseriti**, questo viene risolto sia a livello client che server attraverso un controllo sugli argomenti molto stretto.
Comprende sia l'impossibilità di inserire più codici (-w, -r oppure -l), inserire un file invece che una directory se si usa -l, address non valido o simili.