Progetto Sistemi Operativi e Laboratorio 2021/2022

Elena Cesari

Scelte di implementazione

Le scelte di implementazione sono state effettuate partendo dalla base delineata dalle specifiche del progetto, considerando poi varie opzioni.
Andrò a spiegare le principali scelte fatte per meglio comprendere il funzionamento sia lato server che lato client.

Protocollo di comunicazione

La prima scelta è stata presa per rispondere all'esigenza di avere comunicazioni chiare e congrue tra processo server e processi client; per ottenere ciò sono state stabilite delle convenzioni, le quali sono condivise fra i processi tramite un file header incluso sia nel client che nel server. In questo header sono definiti codici per identificare il tipo di operazione richiesta, codici per identificare l'esito di un'operazione, massimali delle grandezze supportate per file, buffer e stringhe ed infine un carattere per specificare la fine di una comunicazione.

Struttura dati

La seconda scelta principale ha riguardato la struttura dati da utilizzare per lo storage: la scelta è ricaduta su liste sia singolarmente che doppiamente collegate. Le liste presentano svariati vantaggi per cui sono state adottate, tra cui il maggiore è prestarsi bene alla politica FIFO per il rimpiazzamento di un file. Secondariamente vengono usate anche per gestire altri aspetti del progetto, tra cui la lista di richieste al server e alcune liste secondarie usate in fase di cleanup (come ad esempio la lista di socket da chiudere).
Le liste permettono inoltre una facile stesura e manutenzione del codice.
Sono state implementate tramite una piccola libreria statica, in cui vengono definiti vari tipi di liste (di stringhe, di integer e di nodi che contengono puntatori a file e campi con informazioni correlate) e le funzioni e le procedure che vengono usate su di essi.

Avvio server

Il server si avvia leggendo la configurazione contenuta nel file config.txt, il quale viene parsato tenendo conto della seguente struttura prestabilita:
Numero_thread;
Dimensione_storage;
Capacità_Mbytes;
MAXLunghezza_coda_richieste;
Nome_File_Logging;

Gestione richieste

Le connessioni al server vengono gestite sempre dal main thread, il quale si occupa di accettarle e chiuderle. Questo continua poi a monitorarle in un loop, in cui controlla nuove connessioni, richieste da connessioni già presenti e chiusure, tramite la poll.
Le richieste dei client sono servite con una politica FIFO: il main thread mette in coda il client con una richiesta, il primo worker disponibile poi preleverà il nodo con l'fd del client dalla coda e leggerà la richiesta sulla socket.
Per evitare che i worker facciano giri a vuoto sprecando risorse in caso non ci siano richieste da servire, essi vengono messi in stato di wait su una condition-variable, sulla quale il main manda un segnale risvegliando un worker quando arriva una richiesta.
Sulla stessa condition-variable viene eseguito un broadcast alla chiusura del server, in concomitanza al settaggio di una flag, per svegliare i thread e segnalare loro di eseguire un exit per rendersi joinable dal codice di cleanup.

Gestione segnali

Sempre a lato server i segnali sono stati poi implementati creando un apposito thread.
Il thread main pone una maschera, che viene poi ereditata dai worker, evitando che altri thread intercettino dei segnali.
Il thread handler nel mentre entrerà in un loop in cui rimarrà in attesa fino a che non riceve un segnale; alla ricezione setta una flag per indicare il segnale arrivato a tutto il server, esce dal loop ed esegue un exit, evitando quindi l'eventuale arrivo di segnali conflittuali.
Questo è stato fatto soprattutto per evitare che i segnali interrompessero chiamate di sistema, in particolare il main thread avrebbe riscontrato problemi sulla chiamata a poll.
I segnali da considerare erano sighup, sigiq e sigint.
Nel caso di arrivo di sighup prima della chiusura del server era richiesto che venissero servite tutte le richieste dei client attualmente connessi, senza accettare nuove connessioni. Per fare ciò il thread main tiene un contatore delle connessioni chiuse. Non accettando nuove connessioni, quando arriva sighup questo viene incrementato per ogni connessione chiusa. Il server esce dal loop quando, confrontando il contatore con il totale delle connessioni monitorate, tutte le connessioni sono state chiuse; a questo punto setta la flag di segnale per i worker e avvia la chiusura totale del server.

Se invece il segnale è sigint o sigiq il server si chiude il prima possibile: esce quindi immediatamente dal ciclo, segnalando ai workers la chiusura.

Struttura logging

Il server genera un file di logging, in cui registra tutte le operazioni svolte con le informazioni rilevanti riguardo ognuna.
Le operazioni su file vengono registrate con struttura:
[Operazione] [nome_file]: Bytes [grandezza_file]: Process [client_fd]: Res [risultato_op]: Tid [thread_ID]
Vengono registrate anche apertura e chiusura di connessioni.
Inoltre, nella prima riga del file di logging, viene registrata la configurazione con cui è stato avviato il server, ed alla propria creazione ogni thread, main incluso, registra il proprio TID. Questo viene fatto per facilitare il sunto delle statistiche da parte dello script in bash statistiche.sh.

Connessione client

Per quanto riguarda il client è stata presa la scelta di aprire la connessione al server al momento in cui viene parsata l'opzione -f; in caso prima di questa non sia stata incontrata l'opzione -t, verrà scansita tutta la stringa di opzioni, e il tempo verrà impostato come specificato dalla prima apparizione di -t se trovata, altrimenti msec prenderà 0 come valore di default.
Per ogni operazione richiesta prima dell'avvio della connessione viene stampato a terminale un warning, il quale avvisa l'utente dell'impossibilità di portare a termine la richiesta per mancata connessione.

Comunicazione

La comunicazione con il server avviene esclusivamente tramite funzioni specificate nell'api del client.
Per assicurare che venisse aspettato il tempo richiesto e che le comunicazioni avvenissero con il modello richiesta-risposta, nell'api viene utilizzata una funzione interna, la quale invia al server il contenuto di un buffer passatogli con taglia specificata, aspetta msec e successivamente legge la risposta del server.
Per snellire la richiesta di risorse quando il client non aspetta come risposta un file, ma solo un codice di esito, al buffer per la risposta del server verrà allocata una taglia minore; l'allocazione verrà fatta dalla funzione che si occupa di inviare la richiesta specifica, e questa procedura fa parte delle convenzioni stabilite tra server e client.
Tale procedura prevede che per l'invio di un file, sia da client a server che viceversa, venga inviato un header prima del contenuto, nel quale il recipiente si aspetterà di trovare la grandezza del file che sta per ricevere, in modo da limitare sprechi di memoria.

Scrittura file

Era richiesto che la funzione writeFile avesse successo solo se fosse avvenuta in precedenza la creazione del file; a tale scopo writeFile al suo interno chiama la creazione del file, per assicurarsi che esso esista, e solo in seguito se questa ha successo scrive il contenuto del file tramite la funzione appendToFile.
Per i file pubblici è stato deciso di non permetterne la sovrascrittura, per assicurarne l'integrità. In caso quindi un utente voglia riscrivere un file il quale è già presente nel server, dovrà prima chiederne la rimozione ed in seguito la scrittura.

Espulsione file

In caso di ricevimento di espulsione di un file, verrà prima inviato il codice legato all'espulsione, e dopo un consenso da parte del client il file espulso.
Da notare che per questioni di sicurezza il server controlla che il client in questione abbia i permessi necessari per leggere il file da espellere; in caso non li abbia, viene chiesto al client di aspettare che il server elimini il file senza inviarglielo, per proseguire poi con lo svolgimento della richiesta originale.
L'espulsione può avvenire con due richieste: la creazione di un file ancora vuoto (privato o pubblico che sia) in caso questo sia di troppo rispetto alla capacità del server, ed alla scrittura in append di un file, in caso la taglia di questo faccia sforare lo storage in Mbytes del server.

Test

Sono stati creati tre test per mostrare la funzionalità del progetto.
Ad ognuno è stata dedicata una sottocartella, per poterli isolare.
Sono avviabili dalla cartella principale del progetto tramite comando makefile:
make test1, make test2 e make test3.
Non è necessario altri comandi make prima di essi.
Al loro termine è possibile pulire l'ambiente per ripeterli tramite comandi:
make cleantest1, make cleantest2, make cleantest3 o alternativamente make cleantestAll per pulire tutte tre le sottocartelle dei test con un comando unico.
In caso non si vogliano eseguire i test, ma unicamente compilare il progetto il comando da eseguire è make all, e per la sua pulizia make cleanall.
Lo script statistiche.sh quando avviato stamperà su terminale il sunto delle operazioni svolte dal server. Esso prende come argomento la cartella in cui trovare il file log.txt.

Documentazione

Lo sviluppo del progetto è stato documentato in una repository pubblica gitHub.
Questa è consultabile al seguente indirizzo: https://github.com/AtraNeria/SOL_A_project