ISSUES

Problemi nella soluzione proposta

di Lorenzo Casavecchia **<** lnzcsv@gmail.com **>**

Problemi architetturali

1. L'idea di suddividere logicamente e inviare un file in porzioni della stessa taglia non è utilizzata al meglio in quanto lo scambio di contenuti tra client e server avviene in modo sequenziale
   L'attore in trasmissione inizia l'invio del contenuto a partire dalla sua prima porzione, seguita dalla seconda, la terza, e così via fino al riempimento della finestra di spedizione

   L'attore in ricezione si aspetterà di ricevere le prime porzioni del contenuto fino al riempimento della finestra di ricezione
   Da qui in poi, per ambi attori, il trasferimento del contenuto avverrà sempre trasmettendo / ricevendo le porzioni corrispondenti alla finestra di trasmissione / ricezione successiva alla precedente

   Sulla base di queste osservazioni si può notare come non sia possibile

   - richiedere specifiche porzioni del contenuto in questione e ignorarne altre
   - interrompere momentaneamente il trasferimento del contenuto e poi riavviarlo in un secondo momento (in quanto richiederebbe di eseguire da capo la sequenza di messaggi per invio e riscontro di tutte le porzioni già ottenute)

   Le modifiche che potrebbero permettere l'implementazione di questi servizi sono
   - il riscontro delle porzioni di un contenuto ricevute utilizzando NACK invece degli ACK
   - una gestione delle finestre e buffer di spedizione / ricezione più dinamica (che permetta il caricamento nei buffer di un sottoinsieme delle porzioni del contenuto logicamente vicine a quella non riscontrata)
   - un meccanismo di logging dello stato del trasferimento del contenuto (quindi uno storico delle porzioni del contenuto in questione correttamente o non ricevute / riscontrate)
2. La gestione concorrente di più client (lato server) o di più richieste (lato client) basata su multi-processamento non è scalabile
   La soluzione proposta prevede che un client possa generare concorrentemente non più di

N richieste e che il server possa gestire concorrentemente non più di M client e per ciascuna delle N o M richieste venga generato un processo

Per valori di N o M arbitrariamente grandi questo porterebbe ad un numero di processi ugualmente grande che potrebbero in un qualsiasi istante risiedere in CPU

Se il tempo di attesa per risiedere in CPU dovessere essere sufficentemente grande l'altro attore potrebbe superare il numero di timeout massimi consentivi e terminare per inattività

Un'idea più scalabile per la gestione concorrente di client e server potrebbe consistere nell'instaurare 3 threads

- uno in ricezione delle richieste / risposte verso l'altro attore
- uno di gestione delle richieste
- uno di trasmissione delle risposte / richieste verso l'altro attore

3. Come evidenziato dalle istanze di esecuzione in `DESCRIPTION.md` le varie parti che compongono il sistema non funzionano bene insieme, specialmente in presenza di timeout dinamici
   Le istanze di esecuzione hanno inoltre evidenziato che anche in assenza di perdite vi siano ritrasmissioni e il motivo è che la gestione del timeout non è adeguata e presenta bug

Problemi implementativi

Ad aggiornamento odierno l'applicazione presenta i seguenti bug

- Gli attori in ricezione non inviano riscontri per l'ultima finestra di ricezione
- Processi figlio tendono a rimanere in stato `defunct` anche a seguito del loro riscontro tramite `wait` o `waitpid`
- Gli attori in trasmissione non impostano correttamente i valori dei buffer dell'ultima finestra di spedizione (se N fosse il numero di pacchetti che dovrebbero essere trasmessi, l'attore in trasmissione potrebbe inviare pacchetti fino ad $N+1$ dove l'($N+1$)-esimo pacchetto ha campo `data` vuoto)
- Anche con probabilità di perdita di pacchetti nulla un client potrebbe inviare più volte una stessa richiesta al server prima che quest ultimo abbia modo di rispondergli (ed è sicuramente dovuto dalla gestione dei timeout all'invio della richiesta)
- Nonostante la politica di aggiornamento del timer sia la stessa del client, si osserva che il timer del server tenda ad assestarsi al valore minimo e a restarci (questo bug credo sia responsabile dei valori terrificanti nel `retrans_count` del client e del server)

Ad aggiornamento odierno l'applicazione non presenta le seguenti funzionalità

- Non sono presenti meccanismi di file locking per attori che agiscono su uno stesso file

Infine seppur funzionale l'implementazione offerta non è ottimizzata, presenta molteplici ridondanze e la maggior parte delle variabili utili al funzionamento sono condensate nei file `defs.h`