RELAZIONE RETI DI CALCOLATORI

LUCA PEZZOLLA 0124/002411
FULVIO SERAO 0124/002423
SIMONE MICILLO 0124/002439

Descrizione del progetto:

La traccia richiede l'implementazione di un'applicazione p2p all'interno della quale ogni singolo peer mette a disposizione un numero di procedure maggiore di 2. Ognuno di essi, prima di entrare nella rete, deve registrarsi presso un server inviando le procedure messe a disposizione, accompagnate da una descrizione, dal numero di parametri e dal numero di porta su cui sono disponibili. Il server, quindi, non farà altro che salvare le procedure che gli sono state inviate e metterle a disposizione dei vari peer una volta che questi saranno connessi alla rete. Dall'altro lato, i peer, agiranno da server mettendo a disposizione le proprie procedure o, viceversa, agiranno da client per richiamare ed eseguire le procedure degli altri.

Descrizione e schemi dell'architettura

L'architettura utilizzata nel progetto è composta nello specifico da un unico server principale e da tre peer. L'indirizzo IP scelto per ogni processo è stato quello di localhost/loopback (127.0.0.1) mentre il numero di porta del server è 52000. Il numero di porta dei peer, invece, è a discrezione del singolo utente in fase di esecuzione del programma in quanto deve essere inserito come argomento da riga di comando.

Descrizione e schemi del protocollo applicazione

Una volta avviato il server, i peer potranno a loro volta essere avviati ed invieranno automaticamente le loro procedure a quest'ultimo. Ciò significa, quindi, che nella fase iniziale i singoli processi peer agiranno esclusivamente da client nei confronti del server principale. Successivamente, invece, i peer diventeranno loro stessi dei server sulla porta specificata da riga di comando. Inizialmente, non appena un peer viene lanciato, esso invia le sue procedure contenuto nell'array di struct pacchetto al server attraverso una FullWrite. Dall'altro lato il server leggerà il contenuto con una FullRead e riempirà i relativi campi della sua struct. Successivamente, dopo aver determinato il numero di locazioni che dovrà spedire, invierà questa informazione e aspetterà una conferma di lettura. In ultima istanza, il server invierà le procedure degli altri peer al client. Fatto ciò, il peer rimane connesso al server principale diventando esso stesso un server, nel caso in cui altri processi vogliano connettersi ad esso. A tal punto il peer ha tre opzioni:

1) Connettersi ad un altro peer online per utilizzare una delle funzioni disponibili nella rete. In questo caso ci si connette al peer di cui interessa utilizzare la funzione e gli si inviano, attraverso una FullWrite, i parametri di input richiesti.

2) Richiedere al server l'aggiornamento della lista delle procedure disponibili.

3) Continuare ad agire da server permettendo agli altri peer di connettersi ed usare le sue procedure. In questo caso, il peer è in attesa che un altro processo si connetta ad esso. Se ciò avviene, egli legge la richiesta attraverso una FullRead e utilizzando la funzione della libreria string.h, strcmp, verifica se essa corrisponde ad una delle procedure messe a disposizione. Se ciò è vero, esegue la procedura richiesta e invia attraverso una FullWrite il risultato al peer che ne aveva fatto richiesta.

Dettagli implementativi dei peer

Ogni peer possiede una struct Pacchetto all'interno della quale sono contenute le informazioni relative alle procedure che dovranno essere inviate al server. La struct Pacchetto è utilizzata per dichiarare tre array di struct utilizzati per scopi differenti. Ogni campo della struct è dichiarato come array di char. Il primo array di struct, pacchetto[3], è quello nel quale vengono inserite le informazioni relativamente alle procedure ed è quello che verrà inviato al server tramite la funzione FullWrite. Il secondo array di struct "ricezione[100]" è quello nel quale il peer andrà ad inserire le informazioni relativamente alle procedure messe a disposizione dagli altri processi peer. L'ultima struct "richiesta" viene semplicemente usata nel momento in cui il peer agisce da server. Essa viene utilizzata per salvare al suo interno le informazioni relative alla procedura che un client connesso ha richiesto. Dopo essersi connesso al server, il peer manda le sue procedure attraverso una FullWrite; attende quindi dal server il numero di locazioni da leggere e conferma l'avvenuta ricezione di questo dato. In ultima istanza, se il numero in questione è maggiore di zero chiama la funzione "recv" con la flag "MSG_WAITALL" e attende che tutti i num_locazioni*size_struct bytes siano consegnati; qualora il numero di locazioni fosse uguale a zero stampa a schermo un messaggio che conferma di essere l'unico peer connesso alla rete.

La gestione del peer è avvenuta tramite I/O Multiplex; in questo caso con l'utilizzo di due insiemi fdset: uno per la lettura, uno per la scrittura. Attraverso un controllo sul risultato restituito dalla funzione FD_ISSET riusciamo a capire se c'è qualcosa da scrivere all'i-esimo peer connesso, se c'è qualcosa da leggere sul file descriptor 0 (STDIN), se c'è qualcosa da leggere sul file descriptor associato alla socket creata nel momento in cui si diventa server e, per finire, se c'è qualcosa da leggere sul file descriptor della socket con il server principale. Questo controllo è necessario perché la connessione col server rimarrà aperta fino a quando uno dei due processi non si disconnetterà. Il motivo di tale scelta è dovuto al fatto che nuovi peer potrebbero connettersi al server principale ed inviare le proprie funzioni. Il server, quindi, ha il compito di inviare la lista delle procedure aggiornata ai peer corrispettivi.

Dettagli implementativi del server

Il server principale è implementato mediante I/O Multiplex. Così come il peer, anch'esso possiede una struct Pacchetto, identica nei campi a quella utilizzata dai peer, insieme ad una struct to_send che contiene le locazioni da inviare volta per volta ad ogni peer.

Compito importante del server è infatti quello di inviare agli altri peer le procedure disponibili in quel momento, facendo però attenzione di non inviare le funzioni che loro stessi hanno messo a disposizione. Per far ciò, dopo aver letto il contenuto della struct inviata dal peer ed averla salvata in "storage", il server inserisce all'interno della struct to_send le locazioni che non appartengono al peer a cui inviare le procedure disponibili, incrementando ogni volta un contatore che ci dice il numero di locazioni occupate. Così facendo, il server invia al peer la struct to_send contenente adesso le funzioni degli altri peer presenti in quel momento. Questo passaggio è fondamentale per capire con precisione quante locazioni occorre inviare ad ogni processo che ne fa richiesta. Per far ciò, esso controlla di non star copiando nella struct to_send una funzione disponibile sulla stessa porta messa a disposizione da quest'ultimo e di non star copiando una locazione con campo "porta" vuoto (questo controllo serve per non inviare al peer campi vuoti e che non sono stati ancora riempiti oppure sono stati svuotati).

Attraverso le funzione FD_ISSET, si controlla dapprima se c'è un nuovo peer connesso sull'FD della socket creato inizialmente (per accettare la connessione) e successivamente si controlla il FD di ogni peer i-esimo di cui è stata accetta la connessione. Se c'è qualcosa da leggere sul FD i-esimo possiamo trovarci di fronte a tre casi differenti:

1)Numero di byte letti minore di zero, quindi c'è stato un errore durante la system call FullRead.

2) Numero di byte letti uguale a zero, quindi il peer i-esimo si è disconnesso ed è necessario rimuovere le sue funzioni dall'array storage attraverso la funzione memset. Inoltre, viene effettuata una funzione creata appositamente chiamata "fix_memory" che si occupa di ricompattare la memoria dell'array storage. Tale funzione, sfruttando il numero di elementi cancellati, accoda le locazioni successive agli elementi cancellati a quelle precedenti a

questi ultimi, tramite l'utilizzo di contatori ed indici. Infine, ripulisce le locazioni successive all'ultimo elemento occupato all'interno dello storage.

3)Numero di byte letto maggiore di zero e quindi è necessario inviare al peer i-esimo le funzioni degli altri peer già connessi.

Manuale utente

Dopo aver compilato i sorgenti C, per eseguirli da riga di comando è necessario, nel caso si tratti di un file relativo al peer, di inserire come primo argomento da riga di comando la porta sulla quale il processo potrà essere contattato. Non è necessario nessun parametro da riga di comando nel caso si stia eseguendo il codice del server principale. L'utente non può effettuare alcuna operazione sul server se non quella di visualizzare a schermo i log dei peer (e la loro relativa porta) che si sono connessi o disconnessi ad esso. Nel caso del peer, invece, l'utente inizialmente visualizza la lista delle funzioni disponibili (se ve ne sono), oppure un avviso di essere l'unico processo (peer) connesso. A quel punto, è possibile:

1) Premere "1 + invio" per richiedere la lista delle funzioni aggiornate, per verificare che vi siano nuovi peer connessi al server principale da poter "contattare".

2) Premere il tasto "invio" per agire da client e richiamare una funzione messa a disposizione degli altri peer. Per far ciò, basta inserire solamente l'indice della struttura nel quale la funzione è disponibile e successivamente inserire i parametri di input richiesti distanziandoli eventualmente con uno spazio. Se tutto è andato a buon fine verrà visualizzato a schermo il risultato della procedura richiesta.

3) Non premere nessun tasto e quindi agire come server per mettere a disposizione degli altri peer le proprie funzioni.