DESCRIZIONE DELL'ARCHITETTURA DEL SISTEMA E SCELTE PROGETTUALI EFFETTUATE

Il sistema del server funziona mediante l'utilizzo di una socket di accoglienza, utilizzata per ricevere la richiesta di connessione dal client, ed eventualmente accettarla; ed N socket (dove N è il numero di client connessi) ognuna con lo scopo di eseguire le richieste di ogni client.
Tutte le socket non sono bloccanti.

IMPLEMENTAZIONE

Il codice del server è formato da una funzione main che inizialmente stampa su standard input tutte le porte con i relativi stati (utilizzata o libera), successivamente entra in ciclo infinito di accoglienze, dove ogni volta che viene richiesta una connessione da parte di un client, viene verificato se il numero massimo di client ammissibili è stato superato, in tal caso manderà un codice speciale al client che permetterà di avvisarlo di tale situazione e lo inviterà a connettersi in un altro momento. Per la scelta della porta a cui associare un client viene utilizzato un algoritmo che prevede di assegnare al primo client la prima porta disponibile, e così via. I numeri di porta sono messi a disposizione mediante un puntatore ad un puntatore di char resi disponibili a tutti i processi (parent e child) mediante una porzione di memoria condivisa; al fine di dare la possibilità ad ogni child di comunicare che il client a lui associato ha smesso di funzionare, e di conseguenza aggiornare il numero di porte disponibili. Per ogni client viene creato un child che si occuperà di gestire le sue richieste. Viene utilizzata la libreria <signal.h> per gestire l'eventuale chiusura della socket, in modo tale che: quando il thread main smetterà di funzionare verrà chiusa la connessione ad ogni client connesso. Viene utilizzato un array di dimensione pari alla massima quantità di byte trasmissibili in un singolo messaggio tra client e server per scambiare i messaggi di richiesta; tale buffer viene svuotato ogni volta al fine di garantire coerenza con le richieste. Una volta che il client sceglierà l'operazione desiderata verrà fatto un controllo di comparazione tra il buffer e gli interi associati ai comandi eseguibili.
Nel caso di exit, viene chiusa la connessione e viene resa di nuovo disponibile la porta associata a quel client.
Nel caso di list viene creato un identificatore di file usato per aprire in sola lettura il file "list.txt" contenente tutti i file presenti nel server; tali contenuti vengono messi in un buffer grande abbastanza per contenerli, e successivamente vengono mandati al client mediante la socket creata per far comunicare il child i-esimo con il client i-esimo.
Nel caso di upload il server si prepara a ricevere il file in questo modo:

- Viene ricevuto il nome del file
- Viene ricevuta la sua lunghezza
- Viene calcolato il numero di pacchetti da ricevere
- Apre o crea un file associato al nome passato nella propria directory
- Invoca la funzione UDP_GO_BACK_N_Recive e rimane in attesa di pacchetti che verranno inseriti in una struttura grande quanto il numero di pacchetti da ricevere
    - La funzione ha un ciclo while che dura per l'intera durata di ricezione dei pacchetti
    - Viene fatto un controllo per verificare se il numero di pacchetti è un multiplo della finestra di trasmissione, in caso negativo vengono fatti dei provvedimenti che si basano sul ricevere separatamente gli ultimi offset (resto tra numero di pacchetti diviso la finestra di trasmissione) pacchetti
    - Entra in un ciclo for di durata pari alla finestra di trasmissione ed inizia a ricevere i pacchetti
    - Aspetta tutti i pacchetti della serie
    - Invia un riscontro.
    - In caso viene trasmesso un pacchetto fuori ordine o viene perso un pacchetto, viene inviato l'ack inerente all'ultimo pacchetto ricevuto e viene scartato il pacchetto sbagliato.

- Una volta ricevuti tutti i pacchetti viene esaminata l'intera struttura al fine di ricopiare tutti i contenuti ricevuti sul file di testo precedentemente creato

Nel caso di download il server si prepara ad inviare il file in questo modo:

- Viene inviato il nome del file
- Viene inviata la sua lunghezza
- Viene calcolato il numero di pacchetti da inviare
- Crea una struttura per contenere tutti i pacchetti
- Vengono inseriti i pacchetti nella struttura
- Invoca la funzione UDP_GO_BACK_N_Send al fine di inviare tutti i pacchetti
    - La funzione ha un ciclo for che dura per la dimensione della finestra di trasmissione
    - Invia tutti i pacchetti della serie
    - Attendo un riscontro di tali pacchetti.
    - Ogni volta che viene ricevuto un riscontro inerente al primo pacchetto della serie, la finestra scorre.
- Una volta inviati tutti i pacchetti svuota l'intera struttura

Il codice del client è formato da una funzione main che crea una socket per la connessione al server, ed attende che il server gli dia l'esito della tentata connessione.

Dopo di che mostrerà a schermo su quale porta si è connessa.

Successivamente entrerà in un ciclo while di richieste, dove potrà, mediante l'uso di un array di char di dimensione pari alla massima quantità di bytes scambiabili tra client e server, inviare messaggi di richiesta al server, in particolare:

La funzione List permette al client di rimanere in attesa di un puntatore a char sulla socket, contenente tutti i file disponibili sul server, per poi stamparla a schermo

La funzione Exit permetterà di eliminare la socket da lui creata ed inviare prima di chiuderla un messaggio al server notificandogli tale evento.

Nel caso di upload il client si prepara ad inviare il file in questo modo:

- Viene inviato il nome del file
- Viene inviata la sua lunghezza
- Viene calcolato il numero di pacchetti da inviare
- Crea una struttura per contenere tutti i pacchetti
- Vengono inseriti i pacchetti nella struttura
- Invoca la funzione UDP_GO_BACK_N_Send al fine di inviare tutti i pacchetti
    - La funzione ha un ciclo for che dura per la dimensione della finestra di trasmissione
    - Invia tutti i pacchetti della serie
    - Attendo un riscontro di tali pacchetti.
    - Ogni volta che viene ricevuto un riscontro inerente al primo pacchetto della serie, la finestra scorre.
- Una volta inviati tutti i pacchetti svuota l'intera struttura

Nel caso di download il client si prepara a ricevere il file in questo modo:

- Viene ricevuto il nome del file
- Viene ricevuta la sua lunghezza
- Viene calcolato il numero di pacchetti da ricevere
- Apre o crea un file associato al nome passato nella propria directory
- Invoca la funzione UDP_GO_BACK_N_Recive e rimane in attesa di pacchetti che verranno inseriti in una struttura grande quanto il numero di pacchetti da ricevere
    - La funzione ha un ciclo while che dura per l'intera durata di ricezione dei pacchetti

- Viene fatto un controllo per verificare se il numero di pacchetti è un multiplo della finestra di trasmissione, in caso negativo vengono fatti dei provvedimenti che si basano sul ricevere separatamente gli ultimi offset (resto tra numero di pacchetti diviso la finestra di trasmissione) pacchetti
- Entra in un ciclo for di durata pari alla finestra di trasmissione ed inizia a ricevere i pacchetti
- Aspetta tutti i pacchetti della serie
- Invia un riscontro.
- In caso viene trasmesso un pacchetto fuori ordine o viene perso un pacchetto, viene inviato l'ack inerente all'ultimo pacchetto ricevuto e viene scartato il pacchetto sbagliato.

- Una volta ricevuti tutti i pacchetti viene esaminata l'intera struttura al fine di ricopiare tutti i contenuti ricevuti sul file di testo precedentemente creato

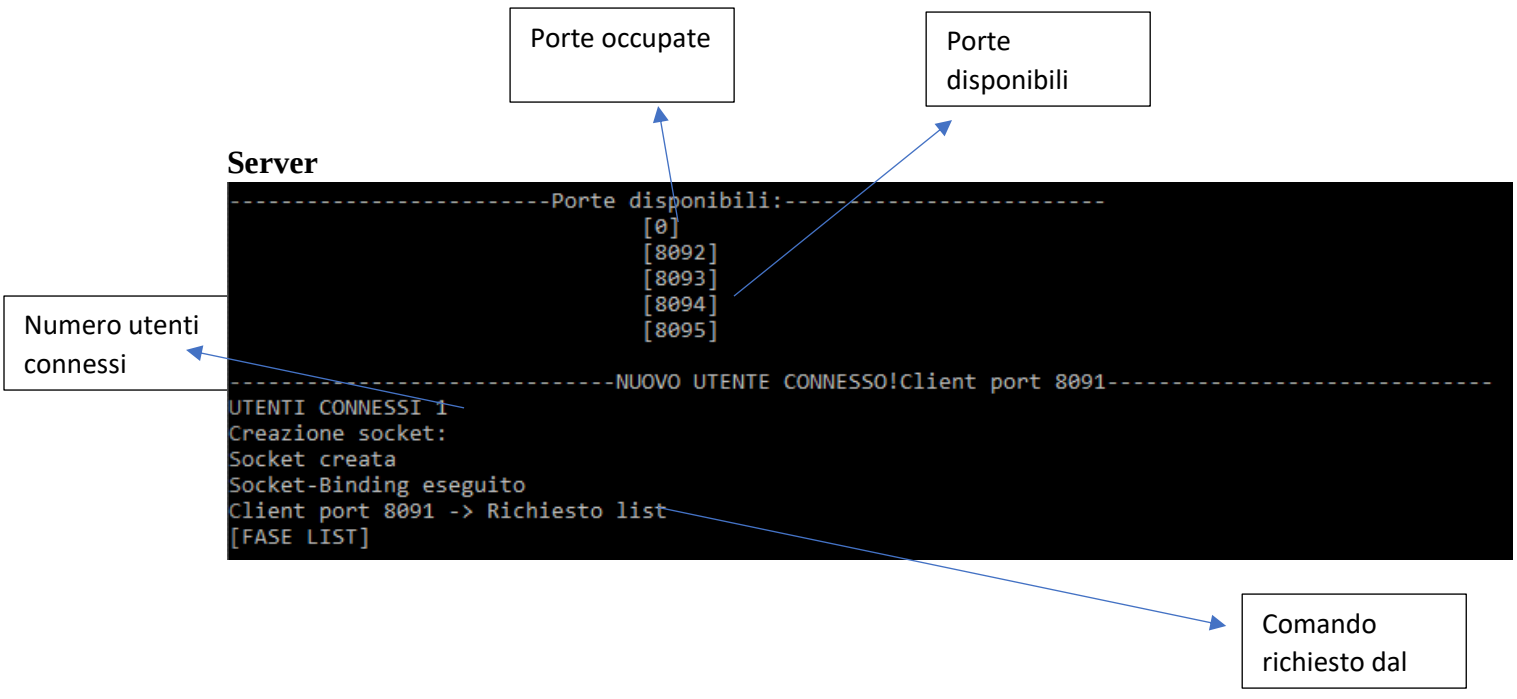
LIMITAZIONI

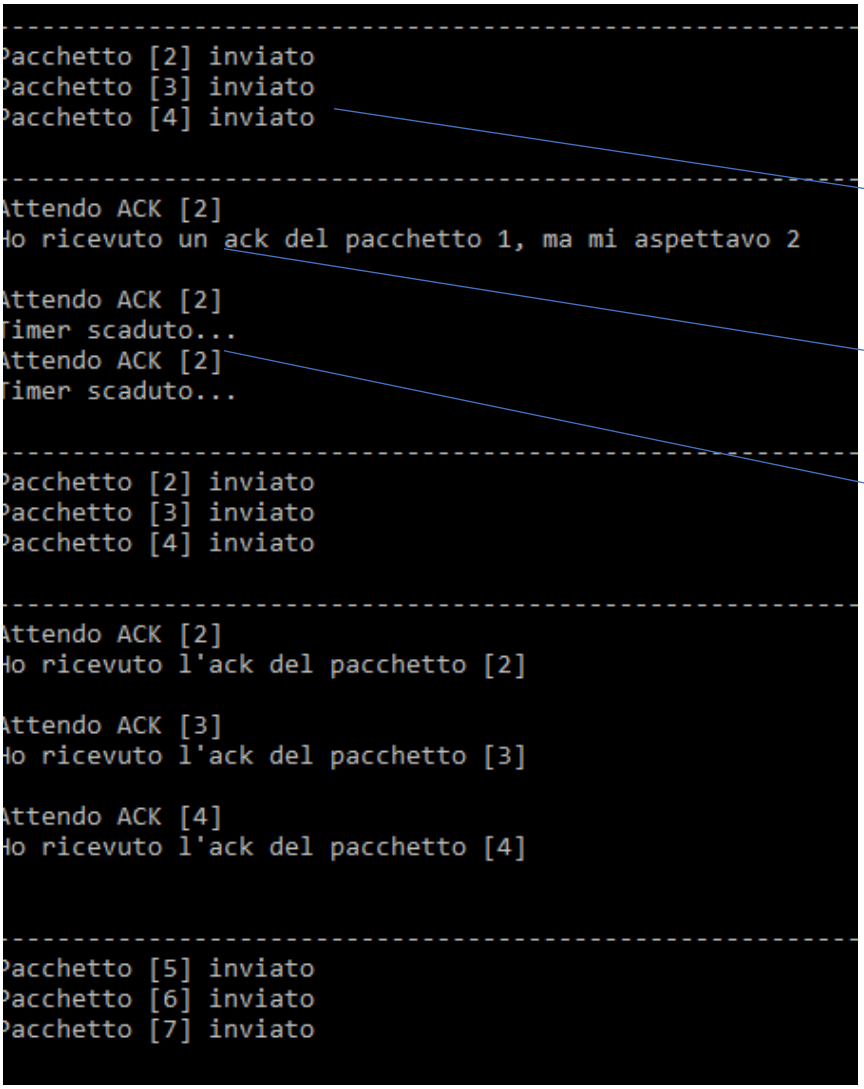
Purtroppo, non sono riuscito ad implementare il timer adattivo…

DELLA PIATTAFORMA SOFTWARE USATA PER LO SVILUPPO ED TESTING DEL SISTEMA

Per lo sviluppo dell'applicazione è stata utilizzata l'applicazione Ubuntu presente su windows, che permette di avere una shell Unix su cui testare continuamente il sistema. Come IDE è stato utilizzato Notepad++

ALCUNI ESEMPI DI FUNZIONAMENTO;

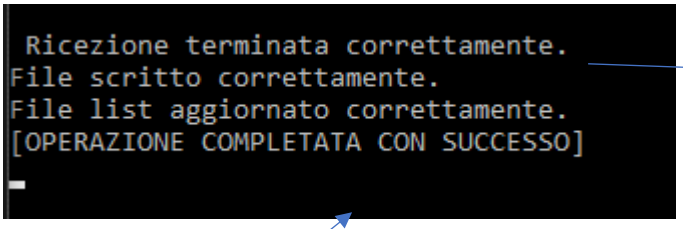




Pacchetti inviati

Simulazione pacchetto

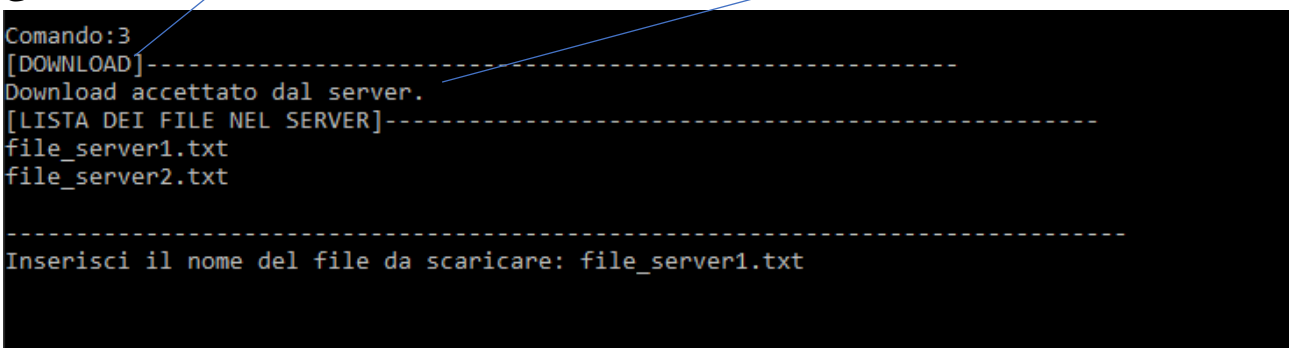
Richiedo ack dell'ultimo pacchetto ancora non riscontrato

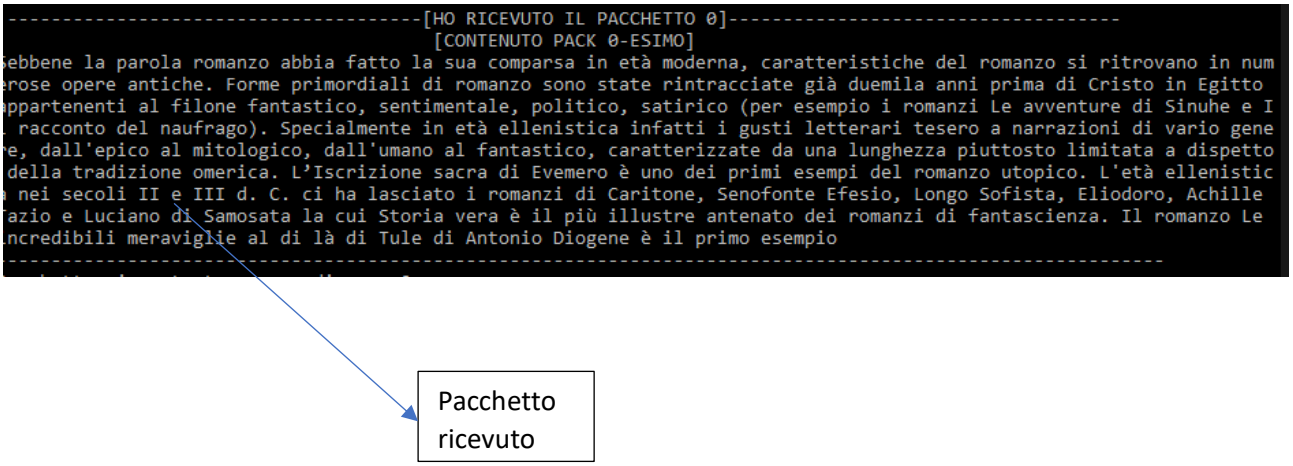


Avviso di completa ricezione

Esito della richiesta

C





VALUTAZIONE DELLE PRESTAZIONI DEL PROTOCOLLO AL VARIARE DELLA DIMENSIONE DELLA FINESTRA DI SPEDIZIONE N, DELLA PROBABILITÀ DI PERDITA DEI MESSAGGI P, E DELLA DURATA DEL TIMEOUT T (INCLUSO IL CASO DI T ADATTATIVO)

FILE DI ESEMPIO:

Client:

- File_client1.txt -> lenght = 1864
- File_client2.txt -> lenght = 10706

Server:

- File_server1.txt -> lenght = 9293
- File_server2.txt -> lenght = 2640

TEMPI DI ESECUZIONE (Media su tre tentativi)

1) Windows size = 3

   Probabilità di perdita = 15% (devo avere un numero inferiore ad 85 su una scala da 0 a 100)

Upload:

- File_client1.txt = 0.038026 s
- File_client2.txt = 0.718414 s
- File_server1.txt = 0.92981 s
- File_server2.txt = 0.152865 s

Download:

- File_client1.txt = 0.12307 s
- File_client2.txt = 1.0141183 s
- File_server1.txt = 1.082798 s
- File_server2.txt = 0.380036 s

2) Windows size = 5

Probabilità di perdita = 15% (devo avere un numero inferiore ad 85 su una scala da 0 a 100)

Upload:

- File_client1.txt = 0.044204 s
- File_client2.txt = 0.67905 s
- File_server1.txt = 0.2294 s
- File_server2.txt = 0.0218 s

Download:

- File_client1.txt = 0.1186 s
- File_client2.txt = 0.2654 s
- File_server1.txt = 0.3564 s
- File_server2.txt = 0.105 s

3) Windows size = 1

Probabilità di perdita = 15% (devo avere un numero inferiore ad 85 su una scala da 0 a 100)

Upload:

- File_client1.txt = 0.042588 s
- File_client2.txt = 0.386085 s
- File_server1.txt = 0.35103 s
- File_server2.txt = 0.060385 s

Download:

- File_client1.txt = 0.25966 s
- File_client2.txt = 0.44775 s
- File_server1.txt = 0.4130 s
- File_server2.txt = 0.11855 s

4) Windows size = 3

Probabilità di perdita = 75% (devo avere un numero inferiore ad 25 su una scala da 0 a 100)

Upload:

- File_client1.txt = 0.03546 s
- File_client2.txt = 6.8615 s
- File_server1.txt = 5.65484 s
- File_server2.txt = 0.02731 s

Download:

- File_client1.txt = 0.0454 s
- File_client2.txt = 6.8256 s
- File_server1.txt = 5.19 s
- File_server2.txt = 0.0765 s

5) Windows size = 5

Probabilità di perdita = 75% (devo avere un numero inferiore ad 25 su una scala da 0 a 100)

Upload:

- File_client1.txt = 0.01339 s
- File_client2.txt = 9.846 s
- File_server1.txt = 8.975 s
- File_server2.txt = 0.09333 s

Download:

- File_client1.txt = 0.0378 s
- File_client2.txt = 9.465 s
- File_server1.txt = 6.865 s
- File_server2.txt = 0.0658 s

6) Windows size = 1

Probabilità di perdita = 75% (devo avere un numero inferiore ad 25 su una scala da 0 a 100)

Upload:

- File_client1.txt = 0.04772 s
- File_client2.txt = 3.892 s
- File_server1.txt = 2.845 s
- File_server2.txt = 0.115 s

Download:

- File_client1.txt = 0.2286 s
- File_client2.txt = 3.957 s
- File_server1.txt = 4.164 s
- File_server2.txt = 0.458 s

MANUALE CONFIGURAZIONE INSTALLAZIONE ED ESECUZIONE DEL SISTEMA

Per installare il sistema Client-Server, basta semplicemente inserire la seguente stringa su terminale Unix:
gcc Server.c -o server
successivamente
gcc Client.c -o client
Per eseguire il sistema bisogna creare una copia del terminale, una per il client ed una per il server.
Nella shell dedicata al server basta inserire la seguente stringa:
. /server
Da quel punto in poi il server metterà a disposizione quali e quante porte d'accesso libere ha a disposizione.
Per far connettere il client al server basta premere semplicemente sulla shell dedicata al client il comando:
. /client

Da quel punto in poi client e server sono in comunicazione; mediante una socket sulla shell del client verranno mostrate le seguenti azioni:

- Exit (per disconnettermi dal server)
- Upload (per caricare un file presente nella directory del client sul server)
- Download (per scaricare un contenuto dal server)
- List (per mostrare i file disponibili sul server)

Tali scelte potranno essere fatte mediante l'inserimento di un numero identificativo dell'azione (visibili in qualsiasi momento dal client)

```
[CONNESSO ALLA PORTA 8091]
Inserisci un comando tra:
1) exit
2) list
3) download
4) upload
```

Ogni scelta effettuata verrà notificata al server mediante una stringa di azione

```
Client port 8091 -> Richiesto list
[FASE LIST]
```

Sulla shell del server verrà continuamente messa a disposizione lo stato delle sue porte per vedere quali client sono attivi, e di conseguenza quanti altri possono accedervi

```
[8091]  [0]
[8092]  [8092]
[8093]  [8093]
[8094]  [8094]
[8095]  [8095]
```

L'intero 0 indica che la porta è attualmente in uso da un client.