Organizzazione di Sistemi Operativi e Reti
Specifiche di Progetto
A.A. 2009/10

Il progetto consiste nello sviluppo di un'applicazione client/server. Client e server devono comunicare tramite socket TCP. Il server deve essere concorrente e la concorrenza deve essere implementata con i thread POSIX. Il thread main deve rimanere perennemente in attesa di nuove connessioni e le deve smistare ad un pool (insieme) di thread che hanno il compito di gestire le richieste.
L'applicazione da sviluppare, *remote-compressor*, deve consentire ad un client di mandare file ad un server per poi farsi restituire un archivio (tar), compresso con un algoritmo di compressione scelto dal client, contenente tali file. A tale scopo devono essere realizzati due programmi, compressor-server dal lato server e compressor-client dal lato client.

Lato client

Il programma compressor-server è un programma interattivo che consente al client:

- la scelta dell'algoritmo di compressione
- la scelta del nome dell'archivio
- la possibilità di visualizzare le configurazioni scelte
- l'invio di uno o più file al server
- la ricezione di un archivio (**tar**) compresso con i file mandati dal client stesso

Il comando per aprire una sessione di lavoro ha la seguente sintassi:

compressor-client <host-remoto> <porta>

dove *host-remoto* è l'indirizzo IP della macchina dove gira il server compressor-server e *porta* è la porta su cui è in ascolto il server. Una volta mandato in esecuzione, compressor-client deve stampare su video un messaggio che informa il client dell'avvenuta connessione e successivamente deve far apparire sullo schermo un prompt (esempio, remote-compressor >). Successivamente l'utente può digitare i comandi per interagire con il server. I comandi che possono essere mandati in esecuzione sono i seguenti:

- **`help`**: tale comando deve mostrare a video una breve guida dei comandi disponibili.
- **`configure-compressor`** `[compressor]`: tale comando deve configurare il server in maniera tale da impostare l'algoritmo di compressione scelto. Gli algoritmi di compressione **devono essere almeno 2**:
  - bzip2
  - gnuzip

  Si riserva allo studente l'eventuale inserimento di ulteriori algoritmi (rar, zip ecc). Una volta eseguito il comando il server dovrà rispondere con un messaggio di successo/errore che verrà visualizzato sul client in modo da permettere all'utente di capire se la configurazione dell'algoritmo di compressione abbia avuto successo o meno.

- **`configure-name`** `[name]:` questo comando deve impostare il nome dell'archivio che vogliamo ricevere. Anche in questo caso il server dovrà rispondere con un messaggio per capire l'esito del comando.
- **`show-configuration:`** tale comando restituisce il nome scelto per l'archivio e il compressore, se tali parametri non sono stati ancora configurati mostra quelli di default.
- **`send`** `[file]`: tale comando prende come parametro il path del file locale che deve essere inviato al server (*local-file*). Per eseguire tale operazione il client deve aprire il file *local-file* e andarne a leggere il contenuto. In caso di errore nell'apertura del file deve essere stampato a video un opportuno messaggio di errore. Una volta letto il file, il client deve inviare al server un comando opportuno fatto seguire dal contenuto del file. Il server prende i dati inviati dal client e li inserisce all'interno del file *remote-file* (mantenere il nome originale). In caso di errore durante la creazione del file, il server deve stampare su video un messaggio di errore e inviare al client un messaggio opportuno. Il client deve ricevere tale messaggio e visualizzarlo. Il file *remote-file* deve essere una copia esatta del file *local-file* (usare l'utility diff per verificare se i due file sono uguali). Nel caso in cui il server non abbia i diritti per eseguire l'operazione richiesta, deve essere spedito al client un messaggio opportuno che lo informa sui problemi incontrati dal server. Questo parametro può essere richiamato n volte per ogni file che vogliamo inserire nell'archivio.
- **`compress`** `[path]:` questo comando crea il targz oppure il tar.bz2 dei file mandati con il comando send e crea il file *compressed-remote-file* con il nome scelto attraverso il comando `configure-name.` Il server dovrà poi aprire tale file, leggerlo, e inviare al client il contenuto. Il server dovrà comunicare al client anche il nome del file. Il client dovrà salvare il file nel *path* specificato. Come per il comando `send` sia server che client dovranno gestire tutte le eccezioni e stampare a video lo stato delle operazioni (successo/errore e tipo). Per esempio se si esegue il comando `compress` senza aver mai eseguito una `send` ovviamente il server dovrà restituire un errore. Se si esegue invece questo comando senza aver eseguito i comandi di `configure-*` il server utilizzerà delle impostazioni di default:
  - **`nome:`** ***archivio***
  - **`compressore:`** ***gnuzip***

  Una volta mandato l'archivio al client il server provvederà ad eliminare tutti i file temporanei.
- **`quit:`** tale comando causa la terminazione della sessione di lavoro iniziata con il comando compressor-client. Più precisamente tale comando chiude il socket con il server ed esce. Il server deve visualizzare un messaggio che attesta la disconnessione del client.

Lato server

Il processo compressor-server rappresenta il server del servizio remote-compressor. Tale processo rimane perennemente in ascolto di richieste di connessione provenienti dai client. Quando un client si connette, compressor-server deve attivare un thread dal pool a cui delegare la gestione del servizio offerto e deve tornare ad attendere altre richieste di connessione. Appena un client si connette, il server deve visualizzare un messaggio che indica che il client con indirizzo IP x.x.x.x si è connesso sulla porta yy. La sintassi del comando compressor-server è la seguente:

compressor-server <porta>
dove *porta* è la porta su cui si deve mettere in ascolto il server. I processi che gestiscono il servizio offerto ai client devono visualizzare tutte le informazioni utili per capire che tipo di operazioni vengono

richieste dai client. Oltre a questo devono anche visualizzare i messaggi di errore di cui abbiamo discusso nei punti precedenti.

Esempio di esecuzione del client:
utente@localhost ~/client $ **./compressor-client 127.0.0.1 4000**
REMOTE COMPRESSOR client, v.0.1
Connected to Server 127.0.0.1 on port 4000
Inserire comandi di lunghezza massima 1024 caratteri!
remote-compressor> **ciao**
CLIENT:comando non valido!
remote-compressor>**help**
I conandi supportati da remote-compressor sono i seguenti:
-) configure-compressor [compressor]
-) configure-name [name]
-) show-configuration
-) send [local-file]
-) compress [path]
-) quit
remote-compressor> **configure-compressor gnuzip**
Compressore configurato correttamente.
remote-compressor> **configure-name archivio_foto**
Nome configurato correttamente.
remote-compressor> **show-configuration**
Nome: archivio_foto
Compressore: gnuzip
remote-compressor>**send foto1.jpg**
File foto1.jpg mandato con successo.
remote-compressor> **send foto2.jpg**
File foto2.jpg mandato con successo.
remote-compressor>**compress .**
Archivio archivo_foto.tar.gz ricevuto con successo.
remote-compressor>**quit**
utente@localhost ~/client $

Esempio di esecuzione del server:
utente@localhost ~/server $ **./compressor-server 4000**
porta 4000
REMOTE-COMPRESSOR server, v.0.1
Attesa di connessioni....
CLIENT 127.0.0.1 connesso
CLIENT 127.0.0.1 eseguito comando help
CLIENT 127.0.0.1 eseguito comando configure-compressor gnuzip
CLIENT 127.0.0.1 eseguito comando configure-name archivio_foto
CLIENT 127.0.0.1 eseguito comando show-configuration
SERVER: ricevuto il file foto1.jpg dal client 127.0.0.1
SERVER: ricevuto il file foto2.jpg dal client 127.0.0.1
SERVER compressione in corso archivio_foto.tar.gz richiesta dal client 127.0.0.1
SERVER: spedito file archivio_foto.tar.gz al client 127.0.0.1
SERVER: client 127.0.0.1 chiude la connessione

Note:

- Client e server si scambiano dei dati tramite socket. Prima che inizi ogni scambio è necessario che il ricevente sappia quanti byte deve leggere dal socket. Ad esempio, quando il client invia al server il contenuto del file da copiare, il server non sa a priori quanti bytes deve ricevere. È

quindi necessario che il client invii al server un'informazione che lo metta in condizione di sapere quando ha letto tutti i bytes che il client gli deve inviare.

- Le specifiche non danno vincoli sulla lunghezza dei parametri inseriti dall'utente tramite tastiera. È tuttavia normale che il client imponga un limite su tale lunghezza. In tal caso, È NECESSARIO che tale vincolo sia espressamente comunicato all'utente (ad esempio, tramite la stampa a video di un messaggio di benvenuto).
- NON È AMMESSO CHE VENGANO INVIATI SU SOCKET NUMERI ARBITRARI DI BYTES. Ad esempio, se l'informazione da inviare server (client) è lunga 5 Bytes, DEVONO ESSERE INVIATI SOLAMENTE 5 BYTES (eventualmente 6 se per qualche motivo sensato ha senso mandare anche il carattere di terminazione stringa)

Valutazione del progetto:

Il progetto viene valutato durante lo svolgimento dell'esame. La valutazione prevede le seguenti fasi:

1. Vengono compilati i sorgenti di client e server (la compilazione avviene con l'opzione -Wall, che segnala tutti i warning. Si consiglia vivamente di usare tale opzione anche durante lo sviluppo del progetto, INTERPRETANDO I MESSAGGI DATI DAL COMPILATORE). Il sistema operativo utilizzato all'esame sarà FreeBSD quindi il progetto dovrà compilare senza errori in questo sistema operativo.
2. Viene eseguita l'applicazione per controllarne il funzionamento rispetto alle specifiche fornite.
3. Vengono esaminati i sorgenti per controllare l'implementazione.