SISTEMI OPERATIVI

CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA ELETTRONICA

SPECIFICHE DI PROGETTO A.A. 2009/2010

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 insieme di thread che hanno il compito di gestire le richieste. L'applicazione da sviluppare è un sistema FTP semplificato. Il server FTP rimane SEMPRE in ascolto di connessioni.
I comandi possibili per l'utente sono:

- lls → local ls.

  Il client stampa a video una stringa informativa tipo: Local files: e a seguito l'elenco dei files/directories contenuti nella directory corrente del client.
  L'output deve essere uguale a quello ottenuto eseguendo ls –lah nella directory di esecuzione del client.
- ls → remote ls.

  Il cliente chiede al server di eseguire la ls –la nella directory di esecuzione del server e inviargli l'output..
  Il client ricevuto l'output lo stampa a video preceduto da un messaggio tipo: Remote files:
- sendFile

  Il client stampa a video la richiesta del nome del file da inviare tipo: File to send:
  Il client aspetta l'input da tastiera. Se il file non esiste viene creato. Il file viene aperto e se l'apertura fallisce viene restituito un errore. Il client invia il file al server se l'apertura non ha generato errore altrimenti stampa a schermo la situazione di errore.
  Il server crea un file con lo stesso nome e vi copia all'interno il contenuto ricevuto dal client.
  Il file creato deve essere una copia esatta!
  Usare diff per verifica che siano uguali.
- receiveFile

  Il client chiede di inserire da tastiera il nome del file da richiedere al server tipo: File to receive:
  Il client aspetta che venga inserito il nome del file da tastiera. Se il file non è contenuto nella directory di esecuzione del server, viene creato dal server e inviato.
  Il server apre il file e se l'apertura fallisce, stampa a video un messaggio e manda al client la notifica di errore.
  In caso di corretta apertura, il server invia il contenuto del file al client.
  Il client crea un file con il nome specificato e vi copia il contenuto mandato dal server.
  Il file creato deve essere una copia esatta!
  Usare diff per verifica che siano uguali.
- quit

  Il client chiude il socket ed esce.
  Il server stampa un messaggio che testimonia la disconnessione del client.

Esempio di esecuzione del server:

```
$ ./ ftp_server
FTP SERVER v.0.1
Waiting for connections…
Client 131.114.5.89 connected!
File list sent to the client
```

```
Received file pippo.c, size 98B
Sent file pluto.c, size 88767B
Client 131.114.5.89 disconnected!
```

Esempio corrispondente sul client:

```
$ ./ ftp_client
FTP CLIENT v.0.5
Connected to server 146.3.6.121
prompt$ ls
Server file list:
...
prompt$ sendFile
prompt$ file name?> pippo.c
Received file pippo.txt, size 98B
prompt$ lls
Local file list:
...
prompt$ receiveFile
prompt$ file name?> pluto.c
Sent file pluto.c, size 88767B
prompt$ quit
$
```

Avvertenze e suggerimenti

- **Test**.
  - un client viene avviato mentre alcuni thread gestori sono già occupati (ma ce n'è almeno uno libero);
  - un client viene avviato mentre non ci sono più thread gestori liberi.
- **Trasferimento dati**
  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. NON È AMMESSO CHE VENGANO INVIATI SU SOCKET NUMERI ARBITRARI DI BYTES.
- **Server**.
  Conviene realizzare il server in modo incrementale, cosicché si possa testare singolarmente il funzionamento di diverse porzioni del codice.
  - Per testare il corretto funzionamento della parte relativa ai socket si può partire da un server che è in grado di servire una sola richiesta alla volta (server mono-processo).
  - Si può estendere il server così ottenuto introducendo la creazione dinamica dei thread (come descritto nei lucidi relativi alla programmazione distribuita).
  - Infine si può modificare il server multi-threaded utilizzando thread preallocati piuttosto che thread creati dinamicamente.

  Ovviamente lo scopo del progetto è creare un server concorrente che utilizzi un pool di thread preallocati.

Valutazione del progetto

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

1. **Compilazione dei sorgenti**. Il client e il server vengono compilati attivando l'opzione `-Wall` che abilita la segnalazione di tutti i warning. Si consiglia vivamente di usare tale opzione anche durante lo sviluppo del progetto, *interpretando i messaggi forniti dal compilatore*.

2. **Esecuzione dell'applicazione**. Il client e il server vengono eseguiti simulando una tipica sessione di utilizzo. In questa fase si verifica il corretto funzionamento dell'applicazione e il rispetto delle specifiche fornite.

3. **Esame del codice sorgente**. Il codice sorgente di client e server viene esaminato per controllarne l'implementazione.