Progetto di Sistemi di Elaborazione

Anno accademico 2023/2024

Il progetto consiste nello sviluppo di un'applicazione client-server denominata **Remote Compressor**, la quale offre un servizio di compressione di file da remoto. L'applicazione deve essere scritta in linguaggio C e funzionare su un sistema Unix o Unix-like.

Il programma client, denominato **`rcomp_client`**, invia uno o più file a un programma server, il quale li aggiunge a un archivio compresso (secondo un algoritmo di compressione scelto dal client) e invia quest'ultimo al client. Il programma server, denominato **`rcomp_server`** si occupa di gestire e comprimere i file ricevuti dal programma client.

La comunicazione tra il programma client e il programma server deve essere realizzata tramite socket TCP, ovvero socket di tipo connection oriented (nota 1).

Di seguito sono riportate le specifiche dettagliate dei programmi client e server dell'applicazione Remote Compressor.

Specifiche del programma `rcomp_client`

Il client deve poter essere eseguito con la seguente sintassi:

```
./rcomp_client [host remoto] [porta]
```

dove `[host remoto]` e `[porta]` sono, rispettivamente, l'indirizzo della macchina e il numero di porta su cui il server accetta richieste di connessione da parte dei client (nota 2). Se non viene passato l'argomento `[porta]`, il client deve utilizzare la porta 1234 come valore predefinito. Se non viene passato l'argomento `[host remoto]`, il client deve utilizzare "127.0.0.1" come valore predefinito (localhost).

All'avvio, il client apre una connessione con il server e, una volta stabilita la connessione, mostra sul video un prompt del tipo `"rcomp> "` in cui l'utente può eseguire i seguenti comandi:

- **`help`**

Mostra a video la lista dei comandi disponibili per il client con una breve descrizione per ciascuno. Esempio di esecuzione:

```
rcomp> help
Comandi disponibili:
help:
 --> mostra l'elenco dei comandi disponibili
add [file]
 --> invia il file specificato al server remoto
compress [alg]
 --> riceve dal server remoto l'archivio compresso secondo
     l'algoritmo specificato
quit
 --> disconnessione
```

- **`add [file]`**

Invia il file di nome `[file]` al server. Si consideri che il file da inviare potrebbe non essere un file di testo. Se il file non esiste, il comando fallisce visualizzando a video un messaggio di errore. Il client deve controllare che l'argomento `[file]` contenga solamente lettere minuscole, maiuscole, cifre e e il carattere `'.'`. Se il nome del file contiene caratteri non inclusi in questa *whitelist*, il file non viene inviato e il client mostra un messaggio di errore.

Esempio di esecuzione 1:

```
rcomp> add a.txt
Invio del file a.txt al server in corso...
File a.txt inviato
```

Esempio di esecuzione 2:

```
rcomp> add b.txt
ERRORE: File b.txt non trovato
```

Esempio di esecuzione 3:

```
rcomp> add ../b.txt
ERRORE: Nome file non valido
```

- **`compress [alg]`**

Dà indicazione al server di creare un nuovo archivio compresso contenente i file precedentemente inviati al server, utilizzando l'algoritmo specificato da `[alg]`. L'argomento `[alg]` può valere `'z'` per l'algoritmo *gzip* oppure `'j'` per l'algoritmo *bzip2*. Se l'utente non specifica alcun parametro, viene utilizzato *gzip* come algoritmo predefinito. Se l'utente usa un valore non valido per `[alg]`, deve essere mostrato un messaggio di errore. Il client riceve l'archivio e lo salva sul file system in un file di nome `archiviocompresso.tar.gz` (se è stato usato l'algoritmo *gzip*) o `archiviocompresso.tar.bz2` (se è stato usato l'algoritmo *bzip2*). I file aggiunti all'archivio devono essere eliminati dal server. Se sul server non erano presenti file, il comando mostra a video un messaggio di errore.

Esempio di esecuzione 1:

```
rcomp> compress z
File archiviocompresso.tar.gz ricevuto
```

Esempio di esecuzione 2:

```
rcomp> compress j
ERRORE: aggiungere almeno un file prima di effettuare la
compressione
```

- **`quit`**

Il client si disconnette dal server e termina l'esecuzione. Nel caso fossero presenti dei file sul server, questi devono essere eliminati.

Esempio di esecuzione:

```
rcomp> quit
Disconnessione effettuata
```

Specifiche del programma `rcomp_server`

Il server deve poter essere eseguito con la seguente sintassi:

```
./rcomp_server [porta]
```

dove `[porta]` è il numero di porta su cui il server accetta richieste di connessione da parte dei client(nota 3). Se non viene passato alcun argomento, il server deve utilizzare la porta 1234.

Lo schema di funzionamento del server è il seguente:

1. Si mette in attesa di connessioni in arrivo da un client;
2. Quando riceve una connessione, la accetta e si mette in attesa di ricevere comandi dal client;
3. Per ogni comando ricevuto dal client, effettua le operazioni corrispondenti:
    - `add`: riceve tramite il socket tutto il file inviato dal client
    - `compress`: riceve tramite il socket il tipo di algoritmo di compressione scelto dal client. Se `'z'`, comprime i file usando l'algoritmo *gzip*; se `'j'`, comprime i file usando l'algoritmo *bzip2* (nota 4). Il nome del file risultante deve contenere solamente lettere minuscole, maiuscole, cifre e e il carattere `'.'`
    - `quit`: chiude la connessione con il client e torna al punto 1

Durante la sua esecuzione il server deve stampare a video delle informazioni descrittive relativamente alle operazioni eseguite (creazione del socket di ascolto, connessioni accettate, operazioni richieste dal client, ecc.).

Un esempio di esecuzione del server è il seguente:

```
$ ./rcomp_server 1234
In attesa sulla porta 1234
Connessione stabilita con il client 127.0.0.1
Ricezione comando add
Ricezione comando add
Ricezione comando add
Ricezione comando compress
Ricezione comando add
Ricezione comando compress
Client disconnesso
In attesa sulla porta 1234
Connessione stabilita con il client 127.0.0.1
Ricezione comando add
...
```

Specifiche generali

- L'eventuale digitazione di `CTRL+C` da parte dell'utente dal programma client deve essere equivalente alla digitazione del comando `quit`.
- Non utilizzare meccanismi di attesa attiva in nessuna parte del codice.
- Mettere in atto delle politiche di gestione delle eventuali situazioni di fallimento delle diverse system call e funzioni di libreria invocate dal programma.
- Utilizzare solamente le system call e le funzioni di libreria viste durante le lezioni/esercitazioni.

Note e suggerimenti

1. Client e server si scambiano dati tramite socket TCP. Prima di ogni scambio di dati tramite il socket TCP, è necessario che il ricevente sappia quanti byte dovrà leggere dal socket.

2. Al fine di testare il programma su una singola macchina, utilizzare l'indirizzo "127.0.0.1" come `[host remoto]`.
3. Assumere che al momento della creazione del socket di ascolto, il programma utilizzi "127.0.0.1" come indirizzo.
4. Sfruttare il programma `tar` per creare un archivio compresso. È possibile eseguire un comando in una shell usando la funzione `system()` (consultare `man 3 system` per maggiori informazioni sul suo utilizzo).

Assumendo che i file sorgenti si chiamino rispettivamente `rcomp_client.c` e `rcomp_server.c`, usare i seguenti comandi per compilare il programma:

- `gcc rcomp_client.c -o rcomp_client`
- `gcc rcomp_server.c -o rcomp_server`

Per testare il programma, aprire due finestre della shell e usarle, rispettivamente, per eseguire un'istanza del programma `rcomp_server` e una istanza del programma `rcomp_client`:

- `./rcomp_server 1234`
- `./rcomp_client 127.0.0.1 1234`

Istruzioni per la consegna

I membri del gruppo devono **inviare una email** a giovanni.nardini@unipi.it e pericle.perazzo@unipi.it indicando nel corpo della mail la volontà di consegnare il progetto e il nome di componenti del gruppo.

I docenti indicheranno una data e un orario in cui fissare un colloquio con il gruppo (in presenza o in modalità remota tramite Teams) per controllare e discutere il progetto svolto.

Importante: l'email deve essere inviata ai docenti **almeno una settimana prima dell'appello** in cui si intende sostenere l'esame. In caso contrario, i docenti non potranno garantire una finestra temporale per lo svolgimento del colloquio.

Il colloquio prevede le seguenti fasi:

1. *Compilazione dei file sorgenti*: il programma deve essere compilato con il comando **gcc** dagli studenti in sede di colloquio su una macchina Unix o Unix-like (ad esempio, la macchina virtuale Debian fornita dai docenti all'inizio del corso);
2. *Esecuzione dell'applicazione*: il programma viene eseguito simulando una tipica sessione di utilizzo al fine di verificare il suo corretto funzionamento e il rispetto delle specifiche fornite;
3. *Discussione sul codice sorgente*: il codice sorgente del programma viene esaminato per controllarne l'implementazione e discutere le scelte implementative adottate dagli studenti.

Tutti i membri del gruppo devono essere presenti al colloquio. Trattandosi di un lavoro di gruppo, è perfettamente lecito che i membri del gruppo si dividano i compiti implementativi. Ciononostante, ciascun membro del gruppo deve essere in grado di spiegare l'implementazione di tutto il codice sorgente e di rispondere alle domande dei docenti su di essa.

L'esito del colloquio può essere di due tipi:

- *Prova superata*: lo svolgimento del progetto è stato giudicato positivamente dai docenti. In questo caso i membri del gruppo potranno presentarsi a uno qualsiasi dei successivi appelli d'esame per sostenere la prova orale. La validità della consegna è permanente.
- *Prova non superata*: lo svolgimento del progetto è stato giudicato non sufficiente (il programma non compila o non esegue, il programma non rispetta le specifiche, l'implementazione è scorretta, ecc.). In questo caso gli studenti dovranno apportare le modifiche richieste dai docenti e richiedere un nuovo appuntamento per la verifica del lavoro.