

Università degli Studi di Pisa
Corso di Laurea in Informatica, Anno Accademico 2001-2002

PROGETTO DEL CORSO DI LPS - A

14 Maggio 2002

1 Introduzione

Il corso di Laboratorio di Programmazione di Sistema prevede lo svolgimento di un progetto finale in gruppi di 2-3 persone¹ per gli studenti che hanno seguito almeno l'80% delle lezioni ed individualmente per tutti gli altri. Il progetto consiste nello sviluppo di software e nella relativa documentazione utilizzando metodi e strumenti forniti durante il corso.

2 Modalità di svolgimento del progetto

2.1 Riferimenti

La pagina Web ufficiale del corso di Laboratorio di Programmazione di Sistema è:

<http://www.di.unipi.it/didadoc/lps>

Da questa, seguendo il link "Corso A" e quindi "Progetto" è possibile accedere, in ogni momento, alle informazioni più aggiornate sul progetto. In particolare, vi vengono riportati: il testo del progetto, l'elenco degli studenti con le frequenze di almeno l'80% delle lezioni, le domande (e le risposte) più frequenti concernenti lo svolgimento del progetto. Si raccomanda, quindi, di consultare il sito con una certa frequenza.

Il docente del corso è reperibile al seguente indirizzo per ogni ulteriore informazione:

Salvatore Ruggieri
Dip. Informatica, Corso Italia 40
Tel. 050 2212782 email:
ruggieri@di.unipi.it
Ricevimento: Martedì 14-17.

¹Non è possibile svolgere il progetto in gruppi di più di 3 persone.

2.2 Tempi e modalità di consegna del progetto

La consegna consiste in una e-mail al docente contenente un file .zip con il software sviluppato e la relativa documentazione. Nella e-mail devono essere indicati i nomi e gli indirizzi e-mail dei componenti il gruppo ed alcune date preferenziali in cui discutere il progetto (lasciando, comunque, almeno una settimana di tempo tra la consegna e la discussione). Il docente risponde alla e-mail fissando una data, possibilmente tra quelle preferenziali.

Il progetto può essere *discusso* in una data qualsiasi delle sessioni di esame entro e non oltre Febbraio 2003 compreso. **Non è possibile discutere il progetto nei periodi di lezione.** Dopo la sessione di Febbraio 2003 il presente progetto *decade*: eventuali consegne successive dovranno riferirsi al progetto dell'A.A. 2002/2003.

2.3 Valutazione del progetto

La discussione consiste nella valutazione del progetto software, ovvero:

- motivazione e originalità delle scelte progettuali,
- strutturazione del codice,
- efficienza e robustezza del software,
- uso di makefile e librerie,
- qualità del codice C,

e della relativa documentazione. La discussione è anche intesa a valutare l'apporto individuale di ciascun componente del gruppo. La valutazione finale è in trentesimi.

Nota 1. Per gli studenti iscritti ai vecchi ordinamenti (lauree quadriennali, quinquennali e diploma), viene assegnata una valutazione del

progetto che concorre alla formazione del voto di Sistemi Operativi I. Le valutazioni sono così determinate:

- A progetto perfettamente funzionante
- B progetto con qualche imperfezione
- C progetto incompleto o non funzionante che comunque a giudizio del docente sia sufficiente
- D progetto non sufficiente.

Nota 2. Per gli studenti iscritti ai vecchi ordinamenti (lauree quadriennali, quinquennali e diploma) che hanno seguito il corso di Laboratorio III precedentemente all'anno accademico 2000-2001, quindi con il programma che prevedeva l'apprendimento del linguaggio di programmazione C, ma che non hanno sostenuto la prova finale, possono sostenerla nel seguente modo:

- svolgono individualmente il progetto di Laboratorio di Programmazione di Sistema Corso A: S. Ruggieri;
- discutono il progetto con i docenti del Laboratorio di Programmazione Concorrente Corso A: C. Bodei, e Programmazione di Sistema Corso A: S. Ruggieri;
- i docenti dei due laboratori concordano il risultato che potrà essere speso per sostenere gli esami di Architetture degli Elaboratori I e Sistemi Operativi I.

3 Il progetto: Mserver

3.1 Introduzione

Lo scopo del progetto consiste nello sviluppo di due programmi in C, *mserver* e *mclient*, per la gestione di messaggi di posta elettronica. Il server viene invocato da un account (d'ora in poi lo chiameremo *Administrator*) che ha i diritti di accesso ad alcuni file dati organizzati come descritto in seguito. Il client può essere invocato da qualsiasi account, ed è una interfaccia tra l'utente ed il server. Il client trasforma comandi dell'utente in richieste al server. Il server accetta richieste dal client e risponde in accordo a quanto richiesto. In ogni momento, esiste al più un server attivo, ma possono esserci zero, uno o più client. La comunicazione dai client al server e viceversa avviene tramite pipe con nome (per cui si assuma pure che *mserver* e *mclient* vengano lanciati sulla stessa macchina).

3.2 Avvio mserver e mclient

Il programma *mserver* risiede nella directory:

```
$HOME/Project
```

dove *\$HOME* è la home dell'account *Administrator*. Tale directory deve avere i seguenti diritti *rwX-----*, cioè solo *Administrator* può eseguire *mserver*. Il programma *mserver* viene avviato da *Administrator* (in una qualsiasi directory) con il comando:

```
> ~/Project/mserver passwdfile
```

dove *passwdfile* è il nome di un file testo con un elenco di identificativi di utenti del server. Ad esempio:

```
> cat passwdfile
pippo
pluto
minni
>
```

All'avvio, il programma *mserver* legge il file *passwdfile* e mantiene in memoria l'elenco dei nomi utente. Il programma *mserver* lavora in background e non termina, a meno di non ricevere un SIGKILL.

Il programma *mclient* può risiedere ovunque e viene avviato da qualsiasi account con il comando:

```
> mclient utente
```

dove *utente* è un nome utente. All'avvio, il *mclient* chiede al *mserver* l'autorizzazione ad operare come *utente*. L'autorizzazione viene concessa dal server se *utente* è nell'elenco di nomi utente memorizzato al suo avvio. Se non viene autorizzato, il *mclient* stampa un messaggio di errore:

```
> mclient paperino
Login Error. Check user name!
>
```

Se viene autorizzato, il client propone un prompt in attesa di comandi dall'utente:

```
> mclient pippo
Login OK.
#      ....  prompt del client in attesa
```

3.3 File dati

I file dati gestiti dal mserver sono messaggi di posta di elettronica. Ogni messaggio è caratterizzato da:

- un unico mittente;
- un unico destinatario;
- un oggetto del messaggio (una riga di testo);
- un testo del messaggio (senza limiti di lunghezza).

`$HOME/Project`² contiene una directory per ogni utente (nel caso una directory per un dato utente non esista, mserver provvede a crearla³). Nella directory di ciascun utente esiste in ciascun istante:

- un file con nome `i.msg`, dove `i` è un numero maggiore uguale a 1; tale file rappresenta l'`i`-esimo messaggio giunto all'utente.

Il messaggio contenuto nel file `i.msg` ha il seguente formato:

- la prima riga contiene il mittente;
- la seconda riga contiene l'oggetto;
- dalla terza in poi c'è il testo.

3.4 Comandi del mclient

Rispetto all'utente, il mclient accetta una serie di comandi interni descritti nel seguito e, analogamente ad una shell, accetta anche comandi esterni (non è richiesto di implementare sequenze di comandi, comandi in background, ridirezione o pipe di comandi). I comandi interni sono: `exit`, `list`, `show`, `get`, `create`.

3.4.1 exit

Il mclient termina.

3.4.2 list

Il comando `list` elenca per tutti i messaggi ricevuti dall'utente: il numero di messaggio, la dimensione in bytes, il relativo mittente ed oggetto del messaggio.

```
# list
numero dim mittente oggetto
1      100 gianni    Ciao
2       85 luigi     Una domanda
#
```

3.4.3 show

Il comando `show c` visualizza il mittente, l'oggetto ed il testo del messaggio numero `c`. Se il messaggio `c` non esiste viene visualizzato un codice di errore.

```
# show 1
from: gianni
subj: Ciao
Un saluto.

Gianni.
# show 4
Error: message number 4 does not exist
#
```

3.4.4 get

Il comando `get` trasferisce nella directory corrente del mclient tutti messaggi ricevuti, eliminandoli sul mserver.

```
# get
# ls
1.msg 2.msg
#
```

3.4.5 create

Il comando `create` permette di comporre un nuovo messaggio indicando destinatario, oggetto e testo del messaggio. Il termine del messaggio è individuato da una riga dove l'utente scrive solo [CTRL-D]⁴. Nell'esempio seguente si riporta in grassetto quanto digitato dall'utente.

```
# create
to: luigi[INVIO]
subj: Saluti[INVIO]
text:
Ciao da Mario[INVIO]
Come stai?.[INVIO]
[CTRL-D]
sent!
#
```

²La chiamata `getenv("HOME")` ritorna il path della home directory.

³Per creare una directory si usi la chiamata `mkdir`.

⁴Si noti che tale occorrenza può essere individuata controllando se una chiamata `read()` ritorna zero.

3.5 Comunicazione mclient-mserver

Il mclient trasforma i comandi utente in richieste al mserver. Tali richieste sono inviate via il pipe con nome `/tmp/clientserver`, il quale, se non esistente, viene creato (con quali diritti?) dal mserver al suo avvio. Si noti che alla terminazione di tutti i mclient, il mserver non deve terminare per effetto della chiusura del pipe in lettura⁵.

Sia *pid* l'identificatore di un processo mclient. In generale, una richiesta consiste di una stringa del tipo:

```
pid CODICE arg ...
```

dove CODICE è una parola che identifica univocamente la richiesta, e arg ... sono gli argomenti della richiesta.

Si noti che non viene ulteriormente specificato come la stringa venga passata dal mclient al mserver, ovvero se prima della stringa viene passata la sua lunghezza, o se la stringa ha una lunghezza fissa, o se la stringa finisce con “\n”.

3.6 Comunicazione mserver-mclient

Il mserver elabora le richieste del mclient (vedi nel seguito) e scrive le risposte nel pipe con nome `/tmp/serverclientpid`, il quale, se non esistente, viene creato (con quali diritti?) dal mclient al suo avvio. Per *pid* si intende l'identificatore del processo mclient. Al termine del mclient il pipe `/tmp/serverclientpid` viene cancellato⁶.

Si noti che non viene ulteriormente specificato come la risposta venga passata dal mserver al mclient, ovvero se prima della risposta viene passata la sua lunghezza, o se la risposta ha una lunghezza fissa, o se la risposta è una stringa che finisce con “\n” o con fine file.

3.7 Elaborazioni del mclient e del mserver

Il mclient trasforma i comandi utente in richieste al mserver. In particolare, le richieste riguardano la autorizzazione ad operare ed i comandi utente:

⁵Un modo per evitare la terminazione è quello di tenere aperto il pipe oltre che in lettura anche in scrittura (senza, però mai scrivere nulla).

⁶La chiamata `unlink(filename)` rimuove il file `filename`.

`list`, `show`, `get`, `create`. Infatti, per il comando `exit` ed i comandi esterni il mclient non ha bisogno di comunicare con il mserver.

Quando il mserver riceve una richiesta, effettua una `fork()`. Il processo figlio esegue le operazioni necessarie a soddisfare la richiesta, quindi termina. Il processo padre rimane in attesa di una nuova richiesta.

3.7.1 Richiesta di autorizzazione

All'avvio, il mclient genera la richiesta sul pipe `/tmp/clientserver`:

```
pid AUTHORIZE utente
```

dove *utente* è l'argomento della riga di comando.

La risposta del (figlio del⁷) mserver sul pipe `/tmp/serverclientpid` consiste della stringa “Login OK.” (il mclient è autorizzato) oppure della stringa “Login Error. Check user name!” (autorizzazione fallita). L'autorizzazione viene concessa solo se *utente* è presente tra quelli memorizzati dal mserver al suo avvio.

3.7.2 list

Il comando `list` genera la richiesta del mclient sul pipe `/tmp/clientserver`:

```
pid LIST utente
```

La risposta del mserver sul pipe `/tmp/serverclientpid` consiste nell'elenco per ciascun messaggio giunto all'utente di: numero messaggio, dimensione, mittente, oggetto.

3.7.3 show

Il comando `show c` genera la richiesta del mclient sul pipe `/tmp/clientserver`:

```
pid SHOW utente c
```

La risposta del mserver sul pipe `/tmp/serverclientpid` consiste di un codice di errore “OK”/”KO”. Nel caso di “OK” segue il mittente, l'oggetto ed il testo del messaggio numero *c* giunto all'utente. Se il messaggio *c* non esiste, al codice di errore “KO” segue “Error: message number *c* does not exist”.

⁷Omettiamo di seguito di specificare tutte le volte che è il figlio a soddisfare le richieste del mclient.

3.7.4 get

Il comando `get` genera la richiesta del mclient sul pipe `/tmp/clientserver`:

```
pid GET utente
```

La risposta del server sul pipe⁸ `/tmp/serverclientpid` consiste di un file `.zip` contenente tutti i messaggi ricevuti dall'utente, i quali vengono eliminati dalla directory del mserver. Il mclient provvede ad decomprimere il file `.zip`, lasciando nella directory corrente i messaggi richiesti. Per le operazioni di compressione/decompressione si veda il manuale dei programmi `zip` e `unzip`.

3.7.5 create

Il comando `create` genera la richiesta del mclient sul pipe `/tmp/clientserver`:

```
pid CREATE utente
```

e quindi l'invio da parte del mclient *su un terzo pipe* `/tmp/clientserverpid` del destinatario, dell'oggetto e del testo del messaggio.

Il mserver provvedere a "spedire" il messaggio al destinatario. Nel caso in cui questo non esista tra gli utenti del mserver, il messaggio verrà spedito al mittente.

4 Requisiti sul codice

- La compilazione del codice avviene mediante un `makefile` appropriato.
- Il codice deve compilare senza errori nè warnings con l'opzione `-Wall`.
- È fortemente consigliato di utilizzare le macro viste a lezione (file `sysmacro.h`) e le funzioni di utilità viste a lezione (file `util.h` e `util.c`).
- NON devono essere utilizzate le seguenti funzioni o chiamate: `printf()` (la `sprintf()` può essere utilizzata); `system()`; funzioni di libreria standard C per l'accesso ai file (peraltro non viste a lezione) quali `fopen()`, `fread()`, `fwrite()`, ecc.

⁸Si presti attenzione a NON passare files per esempio attraverso una directory cui mclient e mserver hanno entrambi accesso. Sebbene abbia lo stesso risultato, questa soluzione è soggetta a problemi di sicurezza (altri processi potrebbero accedere al file durante il passaggio nella directory).

5 Documentazione

La documentazione del progetto consiste dei commenti al codice e di una breve relazione (max 4 pagine) che descriva le principali scelte di progetto.

5.1 Commenti al codice

I file `makefile`, `.c`, `.h` devono essere opportunamente commentati con almeno:

- una intestazione contenente: nome file, nomi autori, specifica del contenuto del file;
- un breve commento prima di ogni funzione che specifichi l'uso della funzione;
- un breve commento per le parti di codice più rilevanti.

5.2 Relazione di progetto

Sono accettati tutti i formati più comuni, quali `.doc` (MS Word), `.txt` (Testo), `.ps` (PostScript), `.pdf` (Adobe Reader) etc. La relazione (max 4 pagine) descrive brevemente:

- strutturazione del codice (`makefile`, `.c`, `.h`) e modalità di compilazione,
- principali scelte di progetto,
- difficoltà incontrate e soluzioni adottate,
- quanto altro si ritiene essenziale.