lkale: una agenda interattiva client-server

Susanna Pelagatti

Progetto di recupero del corso di LPS 2005/06

1 Introduzione

Il progetto consiste nello sviluppo del software relativo ad un sistema client-server che realizza una agenda condivisa fra diversi utenti, e della relativa documentazione utilizzando gli strumenti presentati durante il corso.

1.1 Materiale in linea

Tutto il materiale relativo all'ultimo corso di LPS puó essere reperito su Web alla pagina

`http://www.di.unipi.it/~susanna/LPS/`

Tale pagina conterrà anche le informazioni più aggiornate sul progetto (es. FAQ, suggerimenti, avvisi). Eventuali chiarimenti possono essere richiesti consultando i docenti durante l'orario di ricevimento e/o per posta elettronica.

1.2 Struttura del progetto e tempi di consegna

Il progetto deve essere sviluppato da un singolo studente individualmente e può essere consegnato entro il 1 Febbraio 2007.

La consegna del progetto avviene esclusivamente per posta elettronica, con un file formato TAR che contenga al suo interno la directory `lkale/` che a sua volta contiene (almeno) i seguenti file:

- un file `README` che include una spiegazione dettagliata del contenuto dei vari file e le indicazioni per l'utente (compilazione, installazione, esecuzione, bug noti etc.)
- almeno due file .c:
  - `lkserver.c` che contiene il main del server ed eventuali funzioni ausiliarie
  - `lkclient.c` che contiene il main del client ed eventuali funzioni ausiliarie
- eventuali altri file .c e .h relativi alle librerie sviluppate e ai loro header.
- un file `Makefile` che contenga almeno i seguenti target: `all` che genera gli eseguibili per il server ed il client; e `clean` che elimina i file oggetto (.o)

- la relazione sul lavoro svolto in formato PDF.
- un file `gruppo.txt` che specifica numero di matricola, cognome, nome ed indirizzo di mail dell'autore, secondo il formato (notare la virgola come separatore)

  ```
  Zini, Gino, 123456,  zini@madoc.it
  ```

Si noti inoltre che gli studenti che intendono discutere il progetto entro Luglio 2006 devono consegnarlo entro e non oltre il 30/06/06, gli studenti che intendono discutere il progetto entro Settembre 2006, devono consegnarlo entro e non oltre il 10/09/06, mentre la data ultima di consegna per gli altri è il 01/02/07.

1.3 Valutazione del progetto

Al progetto viene assegnato un punteggio da 0 a 30 in base ai seguenti criteri:

- motivazioni, originalità ed economicità delle scelte progettuali
- strutturazione del codice (suddivisione in moduli, uso di makefile e librerie etc.)
- efficienza e robustezza del software
- aderenza alle specifiche
- qualità del codice C e dei commenti
- chiarezza ed adeguatezza della relazione

La discussione del progetto tenderà a stabilire se lo studente è realmente l'autore e verterà su tutto il programma del corso. Il voto dell'orale (ancora da 0 a 30) fa media con la valutazione del progetto per delineare il voto finale.

Casi particolari Agli studenti iscritti ai vecchi ordinamenti (nei quali il voto di LPS contribuiva al voto di SO) sarà assegnato un voto in trentesimi che verrà combinato con il voto del corso di SO corrispondente secondo modalità da richiedere ai due docenti coinvolti.

2 Il progetto: lkal

Lo scopo del progetto è lo sviluppo di due programmi C, un server *lkserver* ed un client *lkclient* che realizzano una agenda condivisa fra diversi utenti[1] Il server gestisce l'agenda usando un file di testo che ricorda ogni evento con il formato

```
gg/mm/aaaa utente#descrizione
```

[1] Questa è un'estensione del progetto 'lento' proposto per la sessione di Gennaio 2006 di LLS. Gli studenti avessero già svolto tale progetto sono fortemente incoraggiati a riusare il codice sequenziale già sviluppato, eventualmente apportando i miglioramenti suggeriti dal docente di LLS.

in cui il primo campo rappresenta la data dell'evento, l'utente è una stringa di al più 8 caratteri che identifica l'utente che ha registrato l'evento e descrizione costituisce una descrizione sintentica (fino a fine riga).

I client possono richiedere al server di aggiungere nuovi eventi alla rubrica, o di avere informazioni sugli eventi presenti in un certo giorno o mese.

Gli eventi nel file dell'agenda non devono essere mail cancellati (il file serve da memoria storica, e le eventuali correzioni o cancellazioni possono essere effettuate dall'amministratore del server con un qualunque editore di testi).

3 Il server

Il server viene attivato da shell con il comando

```
> lkserver agenda
```

dove `agenda` è il path name del file che contiene l'agenda. Una volta attivato, il server va immediatamente in background (utilizzando la SC fork()). Se `agenda` esiste, lo apre in lettura e scrittura e controlla che il formato sia compatibile con quello descritto nella sezione precedente. Se `agenda` non esiste viene creato. All'attivazione, il server crea anche una pipe con nome fissato

```
lkclientserver
```

dove verranno recapitati i messaggi da parte dei client.

Il server può essere terminato *gentilmente* inviando un segnale di `SIGINT` o `SIGTERM`. All'arrivo di questi segnali il server deve eliminare la pipe di ascolto ed eventuali file temporanei, terminare eventuali scritture in corso nel file `agenda` ed uscire.

Quando il server è attivo accetta dai client richieste di

- aggiunta di un nuovo evento
- richiesta degli eventi in un certo giorno o mese.

Il protocollo di interazione è descritto nella Sezione 5.

4 Il client

Il client è un comando Unix che può essere invocato con diverse modalità

```
lkclient -d gg[/mm/aaaa] -u user descr
  -> inserisce un nuovo evento con descrizione la stringa "descr"
  -> -d specifica la data (se non si specifica
      mese ed anno gg e' il primo prossimo venturo)
  -> -u specifica l'utente che sta effettuando la registrazione
lkclient -g [gg/mm/aaaa]
  -> richiede gli eventi registrati per un certo giorno ed effettua la stampa
     formattata sullo standard output
  -> se il giorno non viene specificato si stampano gli eventi
     registrati nella data odierna
lkclient -m [mm/aaaa]
```

```
  -> richiede gli eventi registrati per un certo mese ed effettua la stampa
     formattata sullo standard output
  -> se il mese non viene specificato si stampano gli eventi
     registrati nel mese odierno
lkclient
  -> stampa un messaggio di uso
```

Segue una descrizione/esemplificazione della varie opzioni. Aggiunta di un evento

```
$$ lkclient -d "07/04/2006" -u chiara "compleanno zia Giulia"
lkclient: Evento
   07/04/2006 chiara#compleanno zia Giulia
registrato!
$$
```

Ecco, invece, un esempio di richiesta giornaliera:

```
$$ lkclient -g 07/04/2006
======================
=  Venerdi 07/04/2006  =
======================
chiara#compleanno zia Giulia
dario#revisione moto
$$$
```

altre formattazioni sono possibili a patto di includere tutte le informazioni riportate sopra.

Per il riepilogo mensile si usa la formattazione 'a calendario'[2] segnalando, in corrispondenza di ciascun giorno solo gli utenti che hanno inserito un evento e (dopo l'asterisco'*') il numero di eventi se maggiore di 1

```
$$ lkclient -m 04/2006
-------------------------------------------------------------------------
|   Lun   |   Mar   |   Mer   |   Gio   |   Ven   |   Sab   |   Dom   |
-------------------------------------------------------------------------
|         |         |         |         |         |    1    |    2    |
|         |         |         |         |         |         |         |
|         |         |         |         |         |         |         |
|         |         |         |         |         |         |         |
-------------------------------------------------------------------------
|    3    |   4     |   5     |    6    |   7     |    8    |   9     |
|         |         |         |         | chiara  |         |         |
|         |         |         |         | dario   |         |         |
|         |         |         |         |         |         |         |
-------------------------------------------------------------------------
|   10    |   11    |    12   |   13    |    14   |    15   |   16    |
|         |         |         |         |         |         |         |
|         |         |         |         | anna*2  |         |         |
|         |         |         |         |         |         |         |
```

[2] La stampa a calendario è la stessa del progetto LLS gennaio 2006.

17	18	19	20	21	22	23
					chiara	gina*2

24	25	26	27	28	29	30

5 Interazione client-server

Server e client interagiscono utilizzando *named pipe*.

Tutti i messaggi inviati dai client verso il server transitano sulla pipe di ascolto del server `./tmp/lkclientserver`[3]. Se tale pipe non esiste viene creata dal server al suo avvio (nb: assicurarsi che alla terminazione di un client il server rimanga attivo!).

All'attivazione, ogni client crea in `./tmp` una pipe dal nome unico (es. `1234` dove 1234 è il pid del client). Questa pipe verrà utilizzata dal server per inviare il messaggio di risposta alla richiesta di un client.

Il formato dei messaggi puó essere liberamente scelto e deve essere documentato e giustificato nella relazione. In particolare sarà necessario prevedere almeno un formato per le inserzioni di eventi da parte del client, un formato per le richieste di informazioni ed i formati per le rispettive risposte da parte del server.

6 Il progetto e la sua documentazione

In questa sezione, vengono riportati alcuni requisiti del codice sviluppato e della relativa documentazione.

6.1 Vincoli sul codice

La stesura del codice deve osservare i seguenti vincoli:

- la compilazione del codice deve avvenire definendo un makefile appropriato;
- il codice deve compilare senza errori o warning utilizzando le opzioni `-Wall -pedantic`
- NON devono essere utilizzate funzioni per la manipolazione delle stringhe che *non* limitano il numero di caratteri scritti/manipolati, ad esempio la `strcpy()` deve essere evitata a favore della `strncpy()` in cui è possibile

[3]Sarebbe più logico creare tutte le pipe di progetto nella directory `/tmp/` di sistema ma per non avere problemi di interazioni indesiderate fra progetti diversi sulle macchine del cli è meglio creare tutte le pipe in una directory `./tmp/` locale alla directory di progetto

fissare il massimo numero di caratteri copiati (per le motivazioni consultare il man in linea)

- NON devono essere utilizzate funzioni di temporizzazioni quali le `sleep()` o le `alarm()` per risolvere problemi di race condition o deadlock fra i processi. Le soluzioni implementate devono necessariamente funzionare qualsiasi sia lo scheduling dei processi coinvolti
- DEVONO essere usati dei nomi significativi per le costanti nel codice (con opportune `#define` o `enum`)

6.2 Formato del codice

Il codice sorgente deve adottare una convenzione di indentazione e commenti chiara e coerente. In particolare deve contenere

- una intestazione per ogni file che contiene: il nome ed il cognome dell'autore, la matricola, il nome del programma; dichiarazione che il programma è, in ogni sua parte, opera originale dell'autore; firma dell'autore.
- un commento all'inizio di ogni funzione che specifichi l'uso della funzione (in modo sintetico), l'algoritmo utilizzato (se significativo), il significato delle variabili passate come parametri, eventuali variabili globali utilizzate, effetti collaterali sui parametri passati per puntatore etc.
- un breve commento che spieghi il significato delle strutture dati e delle variabili globali (se esistono);
- un breve commento per i punti critici o che potrebbero risultare poco chiari alla lettura
- un breve commento all'atto della dichiarazione delle variabili locali, che spieghi l'uso che si intende farne

6.3 Relazione

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 8 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. La relazione *deve rendere comprensibile il lavoro svolto ad un estraneo, senza bisogno di leggere il codice se non per chiarire dettagli implementativi.* In pratica la relazione deve contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali e loro motivazioni)
- la strutturazione del codice (logica della divisione su più file, librerie etc.)
- la struttura del server
- la struttura del client
- l'interazione fra client e server ed i relativi protocolli
- le difficoltà incontrate e le soluzioni adottate

- quanto altro si ritiene essenziale alla comprensione del lavoro svolto
- istruzioni per l'utente su come compilare/eseguire ed utilizzare il codice (in quale directory deve essere attivato, quali sono le assunzioni fatte etc) (da riportare anche nel README)

La relazione deve essere in formato .pdf.