myShell

Andrea Musolino

Dipartimento di Informatica
Università degli studi Sapienza di Roma

Relazione per l'homework del corso di
Sistemi Operativi II
Canale M-Z

Contents

1 Introduzione

L'homework richiedeva di implementare una shell. La shell potrà essere eseguita in due modi:

- **Interactive mode**: verrà visualizzato un prompt e l'utente digiterà un comando in risposta al prompt
- **Batch mode**: viene specificato un file batch in fase di lancio della shell, tale file conterrà i comandi che la shell dovrà eseguire. Non verrà stampato alcun prompt

Sia in modalità interattiva che batch, la shell smette di accettare nuovi comandi quando riceve il comando `quit` su una riga o raggiunge la fine del flusso di input (EOF nel file batch o CTRL-D in modalità interattiva). Ogni riga può contenere più comandi separati da '`;`'. In questa implementazione i comandi verrano eseguiti *concorrentemente*, a differenza del comportamento originale di una shell che li esegue sequenzialmente. Se tra più comandi vi è il comando `quit`, dovranno essere eseguiti, e completati, tutti i comandi che si trovano prima di esso.

2 Scelte progettuali

Come richiesto da specifiche, la shell dovrebbe avere un comportamento base, ovvero:

> *Quando l'utente digita un comando (in risposta al suo prompt), la shell crea un processo figlio che esegue il comando specificato e poi, quando il comando è terminato, restituisce il prompt attendendo un nuovo input. Inoltre la shell deve essere in grado di eseguire più comandi contemporaneamente.*

Per quanto riguarda un comando singolo, ho implementato la funzione `execute_command(char *comm)` il cui comportamento è:

1. Traduce il comando in un array di argomenti
2. Viene controllato se il comando è un comando **built-in**
3. Viene creato un processo figlio, il quale eseguirà il comando passando l'array di argomenti alla funzione `execvp`
4. Il processo padre attende la terminazione del processo figlio

Al passo 2 ho citato *comando built-in*. I comandi built-in, come dice il nome, sono comandi integrati all'interno di una shell, ragion per cui non possono essere eseguiti tramite una funzione della famiglia `exec`. Ho deciso di implementare due comandi built-in (che verrano eseguiti al passo 2 in caso occorrono, tramite la funzione `exec_builtin(char **arguments)`): il comando `cd` per cambiare directory e il comando `quit`, per uscire dalla shell.

Se invece vanno gestiti più comandi (nella linea inserita sarà presente almeno un ';'), viene chiamata la funzione `execute_commands(char *line)`, il cui comportamento è il seguente:

1. Viene controllato se la linea inserita supera il limite o se è mal formattata, in tal caso notifica l'errore su `stderr` senza uscire dalla shell
2. Viene tradotta la linea inserita in un array di stringhe, in cui ogni stringa è un comando
3. Se presente il comando `quit`, viene salvato l'indice che servirà da limite
4. Viene dichiarato un array di threads, la cui lunghezza sarà:
   - $n + 1$, se non è stato inserito il comando `quit`; n è il numero di ';' inseriti
   - `quitIndex`, altrimenti
5. Viene fatto un ciclo su tale array, e per ogni elemento viene chiamata la funzione `pthread_create` per creare un thread, specificando come funzione iniziale `execute_command` che ha il comportamento sopracitato
6. Si attende la terminazione di ogni thread tramite la funzione `pthread_join`
7. Infine se il comando `quit` è stato inserito, si esce dalla shell

Il programma gira attorno queste due funzioni principalmente. Sono state implementatate anche altre funzioni per supportare l'esecuzione di quest'ultime.