

# Esame di Sistemi Operativi

## 1 Traccia

Implementare una programma in linguaggio C che riceva in input, tramite `*argv[]`, il nome di un file  $F$  ed  $n$  stringhe  $s_1, \dots, s_n$  (con  $n \geq 1$ ). Per ogni stringa  $s_i$  dovrà essere attivato un nuovo thread  $T_i$ , che fungerà da gestore della stringa  $s_i$ . Il main thread dovrà leggere indefinitamente stringhe dallo standard-input. **Ogni nuova stringa letta dovrà essere comunicata a tutti i thread**  $T_1, \dots, T_n$  tramite un buffer condiviso, e ciascun thread  $T_i$  dovrà verificare se tale stringa sia uguale alla stringa  $s_i$  da lui gestita. In caso positivo, ogni carattere della stringa immessa dovrà essere sostituito dal carattere `'*'`.

Dopo che i thread  $T_1, \dots, T_n$  hanno analizzato la stringa, ed eventualmente questa sia stata modificata, il main thread dovrà scrivere tale stringa (modificata o non) su una nuova linea del file  $F$ . In altre parole, la sequenza di stringhe provenienti dallo standard-input dovrà essere riportata sul file  $F$  in una forma “*epurata*” delle stringhe  $s_1, \dots, s_n$ , che verranno sostituite da stringhe della stessa lunghezza costituite esclusivamente da sequenze del carattere `'*'`.

Inoltre, qualora già esistente, il file  $F$  dovrà essere troncato (o rigenerato) all’atto del lancio dell’applicazione. Infine, l’applicazione dovrà gestire il segnale `SIGINT` in modo tale che quando il processo venga colpito esso dovrà riversare su standard-output il contenuto corrente del file  $F$ .

## 2 Note sulla soluzione proposta

Due possibili soluzioni sono riportate nel file `prog.c`, infatti sono presenti due implementazioni per la gestione dei segnali richiesti. La prima utilizza la funzione `void (*signal(int, void (*)(int)))(int)`, mentre la seconda utilizza la struttura `struct sigaction` insieme alla system call `int sigaction(int, struct sigaction *, struct sigaction *)`. Sarà sufficiente commentare una delle due porzioni di codice per poter testare l’eseguibile finale.

Mi sono preso la libertà di gestire anche il segnale `SIGQUIT` (l’equivalente di `ctrl+\` indispensabile per la terminazione del programma nel momento in cui gestendo il segnale `SIGINT` il programma non termina), il quale verrà gestito da una routine per la liberazione dei semafori allocati. Ricordo che questo check si può fare durante lo sviluppo del codice utilizzando il comando `ipcs -s`.

Per generare il file eseguibile, è possibile utilizzare il comando `./build.sh` o il comando `./build.sh debug` in base al livello di interazione uomo-macchina che si desidera. Chiaramente le funzionalità in modalità Debug aiutano a capire meglio cosa sta succedendo. Per utilizzare l’eseguibile appena generato è possibile utilizzare il comando `./build/prog` che mostrerà a schermo il seguente output:

```
$ ./build/prog
Usage: ./build/prog <filename> <str-1> ... <str-N>
```

In presenza di problematiche provare ad eseguire il comando `chmod +x build.sh` e nuovamente il comando `./build.sh` o il comando `./build.sh debug`.