

## Architettura dei Calcolatori - Laboratorio 3 - “Captain’s Log”

Gruppo:

Infusini Stefano 5211120

Di Luca Mattia 5213177

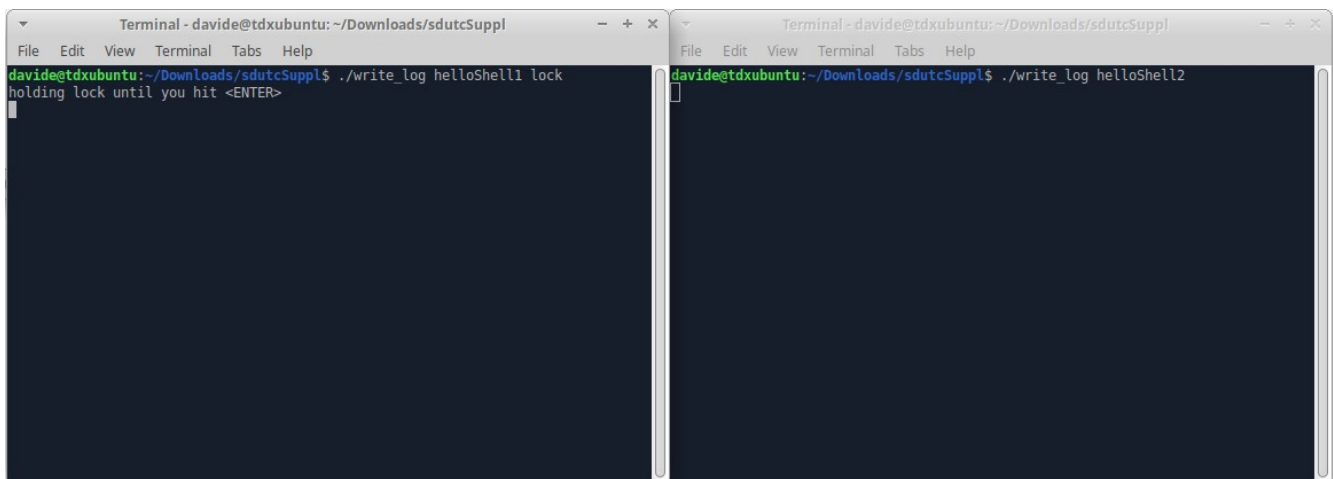
Trioli Davide 5316731

Il programma qui descritto, *write\_log*, effettua la trascrizione di una stringa passata come parametro a linea di comando in un file di testo, un file log, che tiene traccia del momento temporale in cui questa stringa è stata scritta attraverso l'utilizzo di un orario descritto nel formato standard UTC RFC3339.

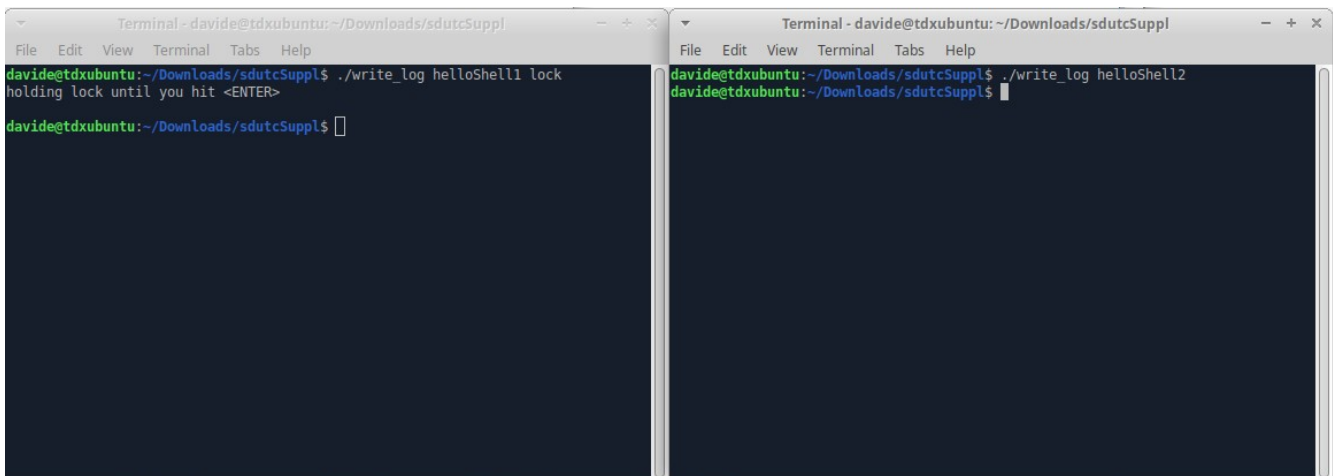
Il programma tiene inoltre in considerazione la possibile sovrapposizione di diversi utenti che potrebbero voler scrivere una propria parola o frase

contemporaneamente sul file di log: per far fronte a questo possibile problema viene utilizzato un sistema di *lock* sul file, in modo che il secondo utente a voler accedere alla risorsa debba prima attendere il rilascio della stessa da parte del primo una volta terminata la sua esecuzione.

Per testare in modo efficiente la correttezza implementativa dei lock sul file, il programma accetta un secondo parametro a linea di comando che permette di non rilasciare il blocco della risorsa fino a un momento deciso arbitrariamente dall'utente: in questo modo è possibile testare con più facilità la corretta esecuzione di quanto descritto.

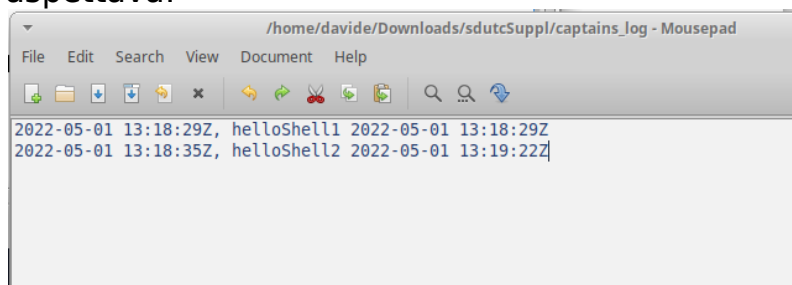


L'immagine sopra descrive la situazione dopo aver inserito i comandi nei terminali, in ordine, 1 e 2: il primo "tiene" il file in lock per una quantità arbitraria di tempo, mentre il secondo non può continuare la sua esecuzione finché non è andato a termine il primo.



Una volta premuto *Enter* nel terminale 1 e quindi avendo portato il programma da questo chiamato alla terminazione, si può notare come anche il secondo abbia potuto continuare e terminare la sua esecuzione.

Il risultato conseguente all'interno del file di log è quindi quello che ci si aspettava:



al suo interno viene prima inserito il primo messaggio e solo successivamente il secondo.

Una ulteriore peculiarità che si può notare nel file di log sono i due orari presenti all'inizio e alla fine di ogni riga, che indicano rispettivamente l'inizio dell'esecuzione del programma e la fine della stessa per una singola sua chiamata.

Si può osservare come, ad esempio, nella prima riga la sua esecuzione è istantanea, in quanto il file era accessibile subito; mentre nella seconda riga la sua terminazione è ritardata: questo è dovuto al mantenimento del blocco relativo all'accesso da parte della prima chiamata, che permette alla seconda un suo accesso solamente posticipato.

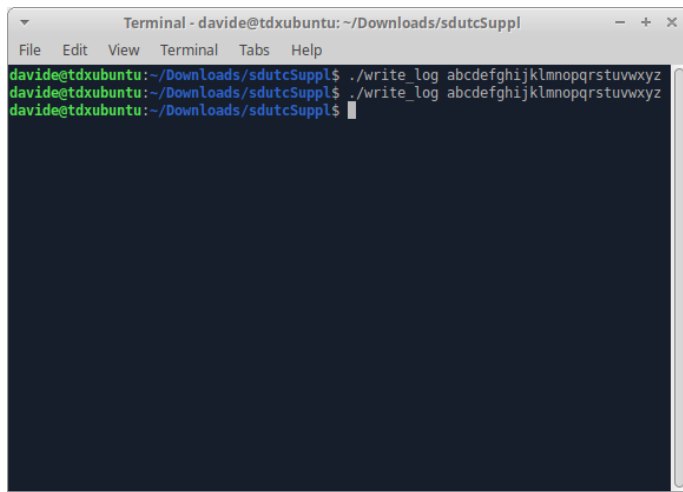
Una ulteriore caratteristica del programma riguarda la lunghezza delle righe scritte all'interno del file: queste, a causa di un buffer di dimensioni limitate, possono contare al massimo di 128 caratteri. La sicurezza relativa alla regolazione di questo buffer avviene in maniera piuttosto semplice all'interno del codice: la stringa passata come parametro, cioè il messaggio personalizzato, viene preventivamente "tagliato" se troppo lungo per essere contenuto nel buffer assieme ai due *timestamps*.

Questa procedura viene effettuata attraverso l'utilizzo di una particolare funzione, ossia *strncat*, come da immagine:

```
tm_sz=20; //20 è il numero di caratteri contenuti in un singolo timestamp
strncat(string,msg,SS-tm_sz-tm_sz-5);
/*
SS è la dimensione massima del buffer.
Oltre a dover conteggiare le dimensioni dei due timestamps, una ulteriore
limitazione di 5 caratteri si rende necessaria per la conformazione della stringa:
',' virgola          1
' ' primo spazio     2
' ' secondo spazio   3
'\n' a capo          4
'\0' fine stringa     5
quindi, del messaggio non possono essere inseriti più caratteri di SS, meno la lunghezza
dei due timestamps, meno ulteriori 5.
Nota: la riga scritta nel file conterrà visivamente 2 caratteri in meno del massimo, in quanto
'\n' e '\0' non vengono visualizzati ma sono comunque contenuti e necessari all'interno del buffer.
*/
```

Un ulteriore test per verificare la correttezza della procedura si può effettuare provando a ridurre la dimensione totale del buffer in modo da rendere più semplice la verifica visiva di quanto descritto.

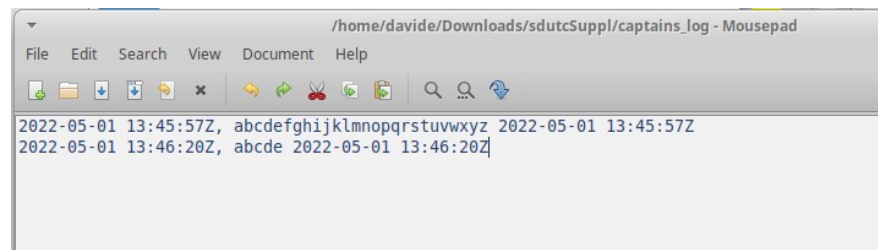
Nell'esempio, il buffer viene ridotto a 50 caratteri massimo mentre si prova a inserire come messaggio l'alfabeto a 26 caratteri: questo verrà necessariamente tagliato in quanto la sua scrittura richiederebbe 40 caratteri per i due timestamps, più 26 caratteri per il messaggio, più gli ulteriori 5 caratteri sopra descritti, per un totale di 71 caratteri che "uscirebbero" in modo erraneo dal buffer.



```
Terminal - davide@tdxubuntu: ~/Downloads/sdutcSuppl
File Edit View Terminal Tabs Help
david@tdxubuntu:~/Downloads/sdutcSuppl$ ./write_log abcdefghijklmnopqrstuvwxyz
david@tdxubuntu:~/Downloads/sdutcSuppl$ ./write_log abcdefghijklmnopqrstuvwxyz
david@tdxubuntu:~/Downloads/sdutcSuppl$
```

Le due chiamate illustrate qui a sinistra si riferiscono a due versioni diverse: nella prima il buffer è quello originario a 128 caratteri, mentre nella seconda è stato ridotto a 50.

L'output prodotto nel file log è quello che ci si attende: la prima chiamata non ha problemi e inserisce il messaggio per intero, mentre nella seconda il buffer ridotto costringe al suo ridimensionamento.



```
/home/davide/Downloads/sdutcSuppl/captains_log - Mousepad
File Edit Search View Document Help
2022-05-01 13:45:57Z, abcdefghijklmnopqrstuvwxyz 2022-05-01 13:45:57Z
2022-05-01 13:46:20Z, abcde 2022-05-01 13:46:20Z
```