Progetto Sistemi Operativi 17/28

Analizzatore Comandi – Logger Statistiche

Indice generale

1. Funzionamento del programma

Con riferimento alle specifiche del progetto, il programma realizzato prende come parametro un comando, e calcola alcune statistiche sulla sua esecuzione, poi le invia ad un processo demone che scrive effettivamente le statistiche in un file di log. Il processo demone deve essere lanciato se non esistente all'inizio dell'esecuzione del programma.

Informazioni salvate dal logger

Il programma esegue il comando e salva alcune statistiche sul suo utilizzo. Se un comando è composto, viene eseguito interamente, ma le informazioni vengono salvate per i singoli sotto-comandi. Ad esempio, inoltrando:

```
stats "ls | wc"
```

I comandi verrò eseguito prima il comando `ls` e poi il comando `wc` con la prevista redirezione dell'output ma nel file di log verranno salvati due record separati.

Le informazioni salvate per ogni comando sono:

- Nome del comando.
- Parametri con cui è stato invocato (compresi di eventuali re-direzioni effettuate tramite gli operatori "`<`" "`>`" "`<<`" "`>>`").
- Tempo di esecuzione: rappresenta il tempo effettivamente speso dall'invio alla terminazione del comando. È ottenuta salvando l'istante di inizio e di fine tramite la funzione `clock_gettime(…)` della libreria `time.h` . Tale funzione viene invocata con l'opzione `CLOCK_REALTIME` per restituire il tempo effettivo. La durata viene espressa in millisecondi.
- Tempo di CPU: tempo di CPU effettivamente speso per l'esecuzione del comando. Il calcolo è analogo a quello effettivo, utilizzando invece l'opzione `CLOCK_PROCESS_CPUTIME_ID` che permette di restituire il tempo di CPU effettivamente dedicato all'intero processo. Viene espresso in microsecondi.
- Codice di ritorno.

Creazione processo demone

Il processo, che diventerà poi demone, prima di tutto crea una coda dei messaggi, una volta che la creazione è andata a buon fine diventa effettivamente un demone.

La “trasformazione” in processo demone viene effettuata utilizzando la funzione `daemon()` della libreria `unistd`. Con riferimento al “Linux Programmer’s Manual” la funzione è utilizzata per staccare un processo dal terminale che lo controlla e essere eseguito in background come un demone di sistema. Per un demone è generato invocando una `fork` , il processo figlio tipicamente richiede un nuovo identificativo di sessione, cambia la directory di lavoro corrente e chiude (oppure redirice a `/dev/null`) tutti i canali di comunicazione standard e temina il padre.

Comunicazione con il processo demone

La comunicazione con il processo demone avviene mediante lo scambio di messaggi con `message queue`.

I messaggi contengono vari campi. Il tipo: rappresenta appunto il tipo di messaggio. I vari tipi di messaggio previsti dal programma sono uno per un messaggio contenente una stringa txt e uno contenente una stringa di log csv e poi un altro tipo che rappresenta un segnale di terminazione per il demone. Gli altri campi sono la stringa di log e il nome del file di destinazione.

Parsing dei comandi

Con riferimento al manuale della shell bash, si specifica come sono stati gestiti i comandi. I comandi, presi in input come vettori di caratteri vengono analizzati, creando una sequenza di cosiddetti “token” i quali rappresentano comandi semplici (sequenze di parole) e operatori. Tale sequenza viene poi analizzata in ordine “da sinistra verso destra”.

Sono gestiti gli operatori:

- and: `&&`
- or `||`
- pipe, sia `|` che `|&`
- sequenze di comandi: `cmd1; cmd2; cmd3;`

Vengono gesti comandi con parametri e con re-indirizzamento dei canali di comunicazione standard.

2. Utilizzo del programma

Il programma può essere nel seguente modo:

```
    stats [opzioni] .. [comando]
```

Dove `[comando]` rappresenta la appunto la stringa del comando da eseguire. Se non formato da una singola parola deve essere racchiuso tra apici. Se viene specificato deve essere inserito come ultimo argomento.

L'argomento [opzioni] opzioni rappresenta il fatto che il comando può essere invocato con alcuni parametri. Tutti le opzioni sono opzionali.

Forma Estesa	**Forma Breve**	**Descrizione**
`--format=[csv | txt]`	`-f=[csv | txt]`	Permette di impostare il formato con cui generare la stringa di log. Utilizzando il valore `csv` i campi saranno separati da virgole, mentre utilizzando `txt` saranno separati da tabulazioni. Non è utilizzabile in combinazione con l'opzione `separator` . Se no specificato il valore di default è `csv`.
`--separator=[string]`	`-s=[string]`	Permette di impostare un separatore non predefinito per separare i campi nella stringa di log. Se non specificato utilizza
`--logfile=[v]`	`-lf=[v]`	Permette di impostare il file di log. Se non specificato utilizza un file di default differente per i log in formato `txt` e `csv`.
`-measure-unit=[true|false]`	`-mu=[true|false]`	Permette di impostare se la stringa di log debba o meno contenere le unità di misura (per i campi in cui sono previste).
`-names`	`-n`	Permette di impostare se la stringa di log debba contenere o meno i nomi dei vari campi.
`--help`	`-h`	Mostra un lungo messaggio di spiegazione del comando.
`--verbose`	`-v`	Se specificato il programma stamperà

		vari messaggi di informazione durante l'esecuzione.
`--usage`	`-u`	Stampa un breve messaggio di informazioni

3. Struttura del Programma

Il programma è stato suddiviso in vari moduli, suddividendo il codice sorgente in più file e directory:

Modulo	Descrizione
`Main`	Rappresenta il punto di ingresso del programma. Si occupa di elaborare gli argomenti passati tramite riga di comando, e di impostare valori adeguati per l'esecuzione del programma.
`Program`	Fornisce la logica di esecuzione del programma. Controlla se esiste il demone che si occupa di scrivere i messaggi di log nei file appropriati e in caso esso non esista lo crea. Fornisce poi agli altri moduli una funzione per l'esecuzione con calcolo delle statistiche di semplici comandi.
`Messanger`	Modulo per astrarre sull'utilizzo delle code per il passaggio di messaggi. Suddiviso in due parti: • `Server`: implementa la logica di messaggistica del demone, ovvero fornisce funzioni per creare ed eliminare coda dei messaggi e per mettersi in ascolto. • `Client`: implementa le funzioni per verificare se il demone esiste e per inviargli messaggi-
`Parser`	Modulo che implementa le funzioni necessarie ad analizzare il comando passato come input. Permette di suddividere ed eseguire comandi composti.
`Execute`	Modulo di appoggio al `Parser`. Una volta che quest'ultimo suddivide un comando in una sequenza di comandi semplici e operatori. Fornisce metodi per l'esecuzione di comandi da semplici a composti, connessi da vari operatori.
`Stats`	Modulo che implementa effettivamente l'esecuzione di un semplice comando e genera una stringa con le statistiche, composta in base alle impostazioni di formattazione.
`Logger`	Modulo che contiene le funzioni per scrivere effettivamente le stringhe di log nel file adeguato.
`Util`	Modulo che contiene funzioni di utilità varie. Contiene la definizione di vari codici di ritorno e costanti utilizzate in varie parti del programma.
`Sys_Wrapper`	Modulo che contiene versioni di chiamate di sistema in cui

	vengono gestiti eventuali errori con una terminazione controllata del programma con stampa di messaggio di errore.