PROGETTO STATISTICHE SHELL

FEATURES

BASE

- Esecuzione comandi
- Poter specificare file di log
- Parametri extra
- Logger indipendente
- Possibile concorrenza tra più shell
- Supporto di qualsiasi comando e carattere speciale
- Statistiche comandi
- Exit Code
- Makefile con regole essenziali

AVANZATE

- Suddivisione di comandi `':'` e `'|'`
- Consistenza id e sub-id comandi
- Ogni sottocomando viene eseguito una sola volta
- Conservazione ambiente shell
- Esecuzione da qualsiasi posizione della shell
- Possibilità di modificare impostazioni

UTILIZZO

MAKE

- `make b`: puliscefile temporanei e compila tutto il necessario
- `make c`: pulisce

ESECUZIONE

- eseguire un comando: è supportato qualsiasi comando, con qualsiasi carattere speciale

  ```
  bash>./bin/run "ls | wc"
  16 16 157
  bash>
  ```

  In assenza di `&&` e `||` il comando verrà spezzato in sottocomandi
- interrompere il logger

```
bash>./bin/run -k true
Logger killed
bash>
```

- cambiare uno o più settings

```
bash@asd:~$ ./bin/run -maxOutput 50 --code=false
Settings updated, logger restarted
bash>
```

- visualizzare gli argomenti possibili

```
bash>./bin/run -h
Settings updated, logger restarted
bash>
```

- altro ??

IMPLEMENTAZIONE

BASE

Esecuzione comandi

Dopo aver suddiviso i comandi in eventuali sotto-comandi, vengono create due pipe: una per inviare comandi alla shell (`toShell`), l'altra (`fromShell`) per riceverne `stdout` e `stderr` . Si effettua un `fork()` . Il child sostituisce `stdin` con `toShell[0]` e `stdout` e `stderr` con `fromShell[1]` (utilizzando `dup2(2)`). Esso eseguirà i comandi attraverso `execvp(2)` . La funzione `executeCommand(5)` può ora inviare comandi a tale child e ricevere i rispettivi output, scrivendo ogni informazione in un `Pk` . Il programma può ora inviare tale `Pk` al logger, sotto formato di stringa (`sendData(2)`), attraverso la fifo `logger.fifo` .

Poter specificare file di log

La variabile .. viene letta all'avvio del logger. Può essere passata come parametro, vedi modifica impostazioni

Parametri extra

Possono essere sfruttati per cambiare le impostazioni/settings. Vengono letti dalla funzione `readArgumets(4)` , poi analizzati da `evaluateCommand(3)` , infine valorizzati da `checkCommandBool(3)` e `checkCommandInt(3)` . Vengono di conseguenza cambiate alcune variabili. Se necessario, il logger viene riavviato.

Logger indipendente

All'avvio del programma, viene controllata l'esistenza del processo di log. Il PID del logger è salvato nel file `logPID.txt`. Attraverso `getpgid(1)` si ottiene lo stato del logger. Se necessario se ne crea uno nuovo effettuando un `fork()` ed eseguendo la funzione `logger(1)`.

Possibile concorrenza tra più shell

Superstringa ??

Supporto di qualsiasi comando e carattere speciale

I comandi vengono eseguiti attraverso `execvp("sh", "sh", "-c", " *comando* ")`, quindi vengono eseguiti correttamente a prescindere dalla loro complessità.

Statistiche comandi

La funzione `executeCommand(4)` ottiene statistiche riguardanti i tempi, oltre ad output ed exit code.

Exit Code

L'exit code si ottiene con l'esecuzione di un `echo $?` subito dopo il comando. Una volta ottenuto tale `int`, viene rimosso dall'output del comando.

Makefile con regole essenziali

Qui deve fare KontortoZ

AVANZATE

Suddivisione di comandi

Viene controllata la presenza di `&&` oppure `||`. In tali casi, nessun comando viene spezzato. Pezzi di comando avvolti da parentesi sono sempre eseguiti in blocco. La divisione avviene in presenza di `|` e di `;`. Il programma spezza il comando in varie parti. Ne esegue una, cattura l'output, in caso di `|` lo inoltra per il sotto-comando successivo, prosegue al prossimo sotto-comando.

Consistenza id e sub-id comandi

Well, `BOH`

Ogni sottocomando viene eseguito una sola volta

Per esempio, alla chiamata di `ls | wc` non vengono chiamati `ls` e poi `ls | wc`. Infatti, potendo mantenere attivo il processo di shell, viene eseguito `ls`. L'output di tale comando viene passato al `wc` attraverso un `echo " *risultato di ls* " | wc`, nel caso in cui il risultato di `ls` sia sullo `stdout`. Se l'output è su `stderr`, il pipe non passa nulla a `wc`, quindi chiamiamo `false | wc`.

Conservazione ambiente shell

Il child che esegue i comandi rimane attivo tra un sottocomando e l'altro, grazie alla possibilità di scrivergli comandi tramite pipe. (forse accennare l' execvp(2))

Esecuzione da qualsiasi posizione della shell

Al momento della compilazione viene creata una macro con la path assoluta del programma. In questo modo anche chiamandolo da fuori della cartella del progetto, i file si troveranno sempre nella cartella del progetto.

Modifica impostazioni

roba

UTILITY

Pack

Lo `struct Pack` oppure `Pk` contiene tutte le informazioni riguardo al comando eseguito:

- tempo di avvio
- tempo di fine
- tempo impiegato
- comando completo
- sotto-comando eseguito
- tipo di output (stdout o stderr)
- output
- return code