Sistemi Operativi, Secondo Modulo
A.A. 2021/2022
Testo del Primo Homework

Igor Melatti

Come si consegna

Il presente documento descrive le specifiche per l'homework 1. Esso consiste in 3 esercizi, per risolvere i quali occorre scrivere 3 files che si dovranno chiamare `1.sh` (soluzione del primo esercizio), `2.sh` (soluzione del secondo esercizio) e `3.awk` (soluzione del terzo esercizio). Per consegnare la soluzione, seguire i seguenti passi:

1. creare una directory chiamata `so2.2021.2022.1.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare `1.sh`, `2.sh` e `3.awk` in `so2.2021.2022.1.matricola`
3. da dentro la directory `so2.2021.2022.1.matricola`, creare il file da sottomettere con il seguente comando: `tar cfz so2.2021.2022.1.matricola.tgz {1..3}.*`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=158` e uploadare il file `so2.2021.2022.1.matricola.tgz` ottenuto al passo precedente.

Come si auto-valuta

Per poter autovalutare il proprio homework, si hanno 2 possibilità:

- usare la macchina virtuale Debian-9 del laboratorio "P. Ercoli" (andando di persona in laboratorio);
- installare VirtualBox (`https://www.virtualbox.org/`), e importare il file OVA scaricabile dall'indirizzo `https://drive.google.com/open?id=1LQORjuidpGGt9UMrRupoY73w_qdAoVCp` (attenzione, sono 9 GB; in caso di difficoltà di download, contattare il docente); maggiori informazioni sono disponibili all'indirizzo `http://twiki.di.uniroma1.it/twiki/view/SO/SO1213AL/SistemiOperativi12CFUModulo220212022#software`. Si

tratta di una macchina virtuale quasi uguale a quella del laboratorio. Si consiglia di configurare la macchina virtuale con NAT per la connessione ad Internet, e di settare una "Shared Folder" (cartella condivisa) per poter facilmente scambiare files tra sistema operativo ospitante e Debian. Ovvero: tramite l'interfaccia di VirtualBox, si sceglie una cartella x sul sistema operativo ospitante, gli si assegna (sempre dall'interfaccia) un nome y ed un mount point (ad esempio, `/mnt/Shared`), e dal prossimo riavvio di VirtualBox sarà possibile accedere alla cartella x del sistema operativo ospitante tramite la cartella `/mnt/Shared` di Debian.

All'interno delle suddette macchine virtuali, scaricare il pacchetto per l'autovalutazione (*grader*) dall'URL `151.100.17.205/download_from_here/so2.grader.1.20212022.tgz` e copiarlo in una directory. All'interno di tale directory, dare il seguente comando:

```
tar xfzp so2.grader.1.20212022.tgz && cd grader.1
```

È ora necessario copiare il file `so2.2021.2022.1.matricola.tgz` descritto sopra dentro alla directory attuale (ovvero, `grader.1`). Dopodiché, è sufficiente lanciare `grader.1.sh` per avere il risultato: senza argomenti, valuterà tutti e 3 gli esercizi, mentre con un argomento pari ad i valuterà solo l'esercizio i (in quest'ultimo caso, è sufficiente che il file `so2.2021.2022.1.matricola.tgz` contenga solo l'esercizio i).

Dopo un'esecuzione del grader, per ogni esercizio $i \in \{1, 2, 3\}$, c'è un'apposita directory `input_output.`i contenente le esecuzioni di test. In particolare, all'interno di ciascuna di tali directory:

- sono presenti dei file `inp_out.`j`.sh` ($j \in \{1, \ldots, 6\}$) che eseguono la soluzione proposta con degli input variabili;
- lo standard output (rispettivamente, error) di tali script è rediretto nel file `inp_out.`j`.sh.out` (rispettivamente, `inp_out.`j`.sh.err`);
- l'input usato da `inp_out.`j`.sh` è nella directory `inp.`j;
- l'output creato dalla soluzione proposta quando lanciata da `inp_out.`j`.sh` è nella directory `out.`j;
- nella directory `check` è presente l'output corretto, con il quale viene confrontato quello prodotto dalla soluzione proposta.

Nota bene: per evitare soluzioni "furbe", le soluzioni corrette nella directory `check` sono riordinate a random dal grader stesso. Pertanto, ad esempio, `out.1` potrebbe dover essere confrontato con `check/out.5`. L'output del grader mostra di volta in volta quali directory vanno confrontate.

Nel seguito, quando si parla di ordinamento lessicografico, si intende quello basato sui byte.

Esercizio 1

Scrivere uno script `1.sh` con la seguente sinossi:

`1.sh [opzioni] [directory...]`

dove le opzioni sono le seguenti (si consiglia l'uso del comando bash `getopts`, vedere `http://wiki.bash-hackers.org/howto/getopts_tutorial`):

- -e string (default: "log"; nel seguito, sia e il valore dato a tale opzione)
- -o string (default: vuoto; nel seguito, sia o il valore dato a tale opzione)
- -h
- -l

L'invocazione dello script è da considerarsi sbagliata nei seguenti casi:

1. viene passata un'opzione non esistente (ovvero, non compresa in quelle elencate sopra);
2. viene passata un'opzione che necessita un argomento, ma senza passare l'argomento;
3. vengono passate sia l'opzione `-l` che l'opzione `-h`.

Se si verifica uno dei casi di errore appena elencati, l'output dovrà consistere nella sola riga, su standard error, `Uso:` p `[-h] [-l] [-e string] [-r regex] [-o file] [dirs]`, con p nome dello script, e lo script dovrà terminare con exit status 10. Nel caso 3, occorre precedere tale scritta con la seguente (sempre su standard error): `Non e' possibile dare contemporaneamente le opzioni -l e -h`.

Nel seguito, siano $d_1, \ldots, d_n$ le directory passate allo script (se non ne viene passata nessuna, assumere $n = 1$ e $d_1 = .$). Occorre controllare le seguenti condizione di errore, assumendo che i file descriptor 3, 4 e 5 siano già aperti da chi chiama `1.sh`.

- Per ogni d_i che non esiste occorre scrivere `L'argomento` d_i `non esiste` su una riga separata sul file descriptor 3; la computazione deve poi continuare ignorando d_i.
- Per ogni d_i che non sia una directory o un link simbolico ad una directory, occorre scrivere `L'argomento` d_i `non e' una directory` su una riga separata sul file descriptor 4; la computazione deve poi continuare ignorando d_i.
- Per ogni d_i che passi i precedenti test, ma non abbia i permessi di scrittura ed esecuzione nei gruppi *group* e *other*, occorre scrivere `I permessi` x `dell'argomento` d_i `non sono quelli richiesti` su una riga separata sul file descriptor 5 (x deve essere il numero ottale corrispondente a permessi e attributi speciali); la computazione deve poi continuare ignorando

d_i. Qualora d_i sia un link ad una directory, contano i permessi della directory puntata.

Lo script deve come prima cosa scrivere su standard output `Eseguito con opzioni` o, dove o è la stringa contenente le opzioni e gli argomenti, come dati da riga di comando. Dopodiché, lo script deve cercare tutti i file $f \in F$ con le seguenti caratteristiche:

- abbiano estensione e (ovvero: il loro nome termini con `.`e);
- si trovino nei sottoalberi di almeno una tra le directory d_i;
- se è stata data l'opzione `-l`, allora bisogna scartare i link simbolici a file;
- se è stata data l'opzione `-h`, allora, per tutti i file che sono hard link allo stesso file, bisogna considerare solo quello con il nome lessicograficamente minore, tenendo conto dell'intero path relativo alla rispettiva directory d_i data come argomento.

Per tutti i file $f \in F$, lo script deve assumere che siano di testo e cercare le righe che abbiano il seguente formato `Execution statistics at` p_i`: system time` s_i`, user time` u_i`, elapsed time` t_i`, memory peak` m_i `kB` (l'indice i sta ad indicare che ogni file può avere più righe così formate). L'output dovrà essere un file CSV di nome o (oppure lo standard output, se o è vuoto) che raccoglie tutte le informazioni dai file in F in questo modo. Tutti i "`time`" sono formattati in uno dei seguenti modi: `d-hh:mm:ss`, `hh:mm:ss`, `mm:ss.ns`, dove `hh, mm, ss` sono ore, minuti e secondi su 2 cifre, `d` è un numero di giorni ed `ns` sono frazioni di secondi, entrambi su un numero imprecisato di cifre. Sia P l'insieme di tutte le diverse stringhe p_i in tutti i file F: molte di queste stringhe saranno ripetute in file diversi, essendo statistiche in determinati punti dello stesso programma. Per ogni $f \in F$, l'output deve contenere una riga formattata come segue: f`,`s_1`,`u_1`,`t_1`,`m_1`,...,`s_k`,`u_k`,`t_k`,`m_k`,`s`,`u`,`t`,`m, dove $k = |P|$, $m = \max_{i=1}^{k} m_i$, $s = \sum_{i=1}^{k} s_i$ (in secondi, rappresentato in virgola mobile con 3 cifre di mantissa) ed analogamente per u e t. Se qualche p_i manca nel file dato, i rispettivi s_i,u_i,t_i,m_i devono essere vuoti. Come intestazione (ovvero, come prima riga), il file di output deve avere la stringa `Filename,`p_1`systime,`p_1`usertime,`p_1`eltime,`p_1`mem,...,`p_k`systime,`p_k`usertime,`p_k`eltime,`p_k`mem,systimetot,usertimetot,eltimetot,memhigh`, ordinando lessicograficamente sugli p_i. Ovviamente, i valori devono essere in corrispondenza con l'intestazione; le ultime 4 colonne contengono le somme (per i tempi) e il massimo per la memoria, considerando ovviamente solo la riga corrente. Inoltre, le righe dell'intero file di output vanno ordinate, dalla seconda riga in poi, nel seguente modo:

- descrescente sui file con numero n di statistiche più alto (quindi, più righe formattate come sopra);
- per i file con uguale valore di n, ordinare crescentemente sull'ultimo memory peak del file;

- se entrambi gli elementi di sopra coincidono, ordinare crescentemente e lessicograficamente sull'intero path relativo del file.

Infine, occorre aggiungere un'ultima riga avente come primo campo `TOTALS`, e per ogni campo i la somma dei tempi della colonna i-esima (se corrisponde ad un tempo) o il massimo delle memorie nella colonna i-esima (altrimenti). Lasciare vuote le ultime 4 colonne di tale ultima riga.

L'exit status dello script deve essere il numero di file o directory, tra gli argomenti dello script, che sono stati ignorati secondo le regole di cui sopra. Inoltre, l'exit status va anche scritto in ottale su standard error. Infine, le righe degli output sui file descriptor 3, 4 e 5 devono essere ordinati lessicograficamente dal più grande al più piccolo e senza righe ripetute.

Attenzione: non è permesso usare Python, Java, Perl o GCC. Lo script non deve scrivere nulla sullo standard error, a meno che non si tratti di uno dei casi descritti esplicitamente sopra. Analogamente, non deve scrivere nulla sullo standard output, tranne che nei casi indicati esplicitamente sopra. Per ogni test definito nella valutazione, lo script dovrà ritornare la soluzione dopo al più 10 minuti.

Esempi

Da dentro la directory `grader.1`, dare il comando `tar xfzp all.tgz input_output.1 && cd input_output.1`. Ci sono 6 esempi di come lo script `1.sh` può essere lanciato, salvati in file con nomi `inp_out.`i`.sh` (con $i \in \{1, \ldots, 6\}$). Per ciascuno di questi script, la directory di input è `inp.`i, e la directory con l'output atteso è `check/out.`i; lo standard output atteso sarà nel file `check/inp_out.`i`.sh.out`, mentre lo standard error atteso sarà nel file `check/inp_out.`i`.sh.err`.

Esercizio 2

Scrivere uno script bash con i seguenti argomenti (nell'ordine dato):

1. intervallo di campionamento c;
2. nome di un file F.

Se uno dei suddetti input manca, l'output dovrà essere semplicemente la scritta `Uso:` s `sampling commandsfile` su standard error (dove s è il nome dello script), e lo script dovrà terminare con exit status 30.

Il file di testo F si può assumere formattato come segue: per ogni riga, ci sono 2 campi separati dal carattere pipe `|`. Il primo campo contiene un nome di un comando C_i, eventualmente con i suoi argomenti; il secondo campo contiene una sequenza di n_i coppie (N_{ij},G_{ij}) tali che N_{ij} è un numero intero e G_{ij} è un nome di un file.

Lo script dovrà lanciare in background e monitorare tutti i comandi C_i presi dal file F. Inoltre, ogni comando C_i dovrà essere rediretto n_i volte, in modo tale che il file descriptor N_{ij} sia rediretto in G_{ij}. Qualora uno dei comandi C_i non esista, non va lanciato. Una volta lanciati tutti i comandi, occorre scrivere sul file descriptor 3 i PID dei processi lanciati (tutti sulla prima riga, separati dal carattere _), *prima* di proseguire con la computazione descritta qui sotto. Lo script, se non riscontra errori, deve rimanere in esecuzione finché non trova un file regolare `done.txt` sulla current working directory. Il monitoraggio di questa condizione deve avvenire ogni c secondi. A quel punto, deve scrivere sul file descriptor 4 `File done.txt trovato`, seguito dalla scrittura su standard output della foresta dei processi e dei thread di C, per poi terminare con exit status 0. La foresta dei processi e dei thread va scritta come una sequenza di righe formattate come segue:

$$p_i(t_{i1} \dots t_{ik_i}) : p_{i1} \dots p_{il_i}$$

dove la riga i-esima ha le seguenti proprietà:

- p_i è il PID del processo risultante dall'esecuzione di un comando in F;
- p_{ij} sono i figli di p_i (in ordine crescente di PID);
- se p_i è diviso in thread, $t_{i1} \dots t_{ik_i}$ sono i PID dei thread che lo compongono, e p_i è il `tgid`; altrimenti, la parte tra parentesi contiene solo il PID del processo stesso.

Le radici della foresta p_i devono corrispondere a tutti e soli i processi C_i. Le righe vanno ordinate (numericamente) sul primo PID. Se invece i processi terminano tutti prima che sia possibile monitorarli per scrivere la foresta, occorre scrivere su standard output `Tutti i processi sono terminati` e uscire con exit status pari ad n (numero dei comandi C_i).

Attenzione: non è permesso usare Python, Java, Perl o GCC. Lo script non deve scrivere nulla sullo standard error, a meno che non si tratti di un errore

nelle opzioni da riga di comando come descritto sopra. Non deve mai scrivere nulla sullo standard output, tranne che nei casi descritti sopra. Per ogni test definito nella valutazione, lo script dovrà ritornare la soluzione dopo al più 10 minuti, e lanciare i comandi passatigli entro al più 3 secondi.

Esempi

Da dentro la directory `grader.1`, dare il comando `tar xfzp all.tgz input_output.2 && cd input_output.2`. Ci sono 6 esempi di come lo script `2.sh` può essere lanciato, salvati in file con nomi `inp_out.`i`.sh` (con $i \in \{1, \ldots, 6\}$). Per ciascuno di questi script, la directory `inp.`i contiene uno script `main.sh` e degli altri file necessari per lanciare `2.sh` e controllare che funzioni correttamente. La directory `check/out.`i contiene i file con l'output atteso; lo standard output atteso sarà nel file `check/inp_out.`i`.sh.out`, mentre lo standard error atteso sarà nel file `check/inp_out.`i`.sh.err`.

Esercizio 3

Un file di log è una sequenza di righe del seguente formato:
T s

dove T è un timestamp e s una stringa che descrive un evento. Il timestamp può avere uno dei seguenti formati:

- d m D H
- m D H
- $Y/M/D$ H
- $Y/M/D$ $H.s$

dove d sono le 3 lettere iniziali del giorno in inglese (`Mon`, `Tue`, etc), m sono le tre lettere iniziali del mese in inglese (`Jan`, `Feb`, etc), D è il giorno del mese in numero di 2 cifre, H son le ore, i minuti ed i secondi, sempre su 2 cifre, separati da `:`, s sono millisecondi, Y è un anno su quattro cifre, M è il mese in numero di 2 cifre. Se l'anno non è presente, assumere che sia il 2022.

Uno script di configurazione I è formato da righe con il seguente formato

variabile=valore

Le righe di I possono seguire un qualsiasi ordine; è possibile che delle righe siano ripetute, nel qual caso conta la prima ad apparire dall'alto. Nel nostro caso, siamo interessati alle seguenti variabili: `from`, `to`, `text`. Siano f, t, x i valori (possibilmente vuoti) che risultano assegnati a tali variabili, rispettivamente. I valori f e t possono avere lo stesso formato di date enunciato sopra, mentre x è un generica stringa.

Lo script `3.awk` dovrà prendere come argomenti un file di configurazione I ed un certo numero di file di log $F_1, \ldots, F_k$, con $k \geq 1$. Quindi, deve scrivere su standard output, come prima cosa, la sequenza dei nome dei file passatigli come argomento, tutti su una riga separati da spazi e preceduti dalla stringa `Eseguito con argomenti`. Dopodiché, dovrà scrivere, sempre su standard output, nell'ordine, tutte le righe tratte da un qualche file di log F_i e aventi la data compresa tra f e t e il testo contenente x. Prima di riportare la riga, va anche scritto il nome del file F_i dove si trova, seguito dai due punti (come fa il comando `grep`). Se uno dei valori di f, t, x è vuoto, si considera il corrispondente test come automaticamente passato: ad esempio, se x è vuoto, allora tutte le righe che soddisfano f e t vanno stampate, se f è vuoto e t no, allora si considerano tutte le righe con timestamp più piccolo di t, etc.

Lo script non deve scrivere nulla sullo standard error, tranne che nel caso in cui non vengano dati almeno 2 file di input; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: dare almeno 2 file di input`, e terminare senza effettuare altre computazioni con exit status 10;

Attenzione: per ogni test definito nella valutazione, lo script dovrà ritornare la soluzione dopo al più 10 minuti.

Esempi

Da dentro la directory `grader.1`, dare il comando `tar xfzp all.tgz input_output.3 && cd input_output.3`. Ci sono 8 esempi di come lo script `3.awk` può essere lanciato, salvati in file con nomi `inp_out.`i`.sh` (con $i \in \{1, \ldots, 6\}$). Per ciascuno di questi script, la directory `inp.`i contiene i file di input. La directory `check/out.`i contiene i file con l'output atteso; lo standard output atteso sarà nel file `check/inp_out.`i`.sh.out`, mentre lo standard error atteso sarà nel file `check/inp_out.`i`.sh.err`.