Sistemi Operativi, Secondo Modulo
A.A. 2020/2021
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 `so2.2020.2021.1.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare `1.sh`, `2.sh` e `3.awk` in `so2.2020.2021.1.matricola`
3. da dentro la directory `so2.2020.2021.1.matricola`, creare il file da sottomettere con il seguente comando: `tar cfz so2.2020.2021.1.matricola.tgz {1..3}.*`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=116` e uploadare il file `so2.2020.2021.1.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" (stanti le ultime limitazioni, questa possibilità è purtroppo da scartare);
- installare VirtualBox (`https://www.virtualbox.org/`), e importare il file OVA scaricabile dall'indirizzo `https://drive.google.com/open?id=1LQORjuidpGGt9UMrRupoY73w_qdAoVCp`; maggiori informazioni sono disponibili all'indirizzo `http://twiki.di.uniroma1.it/twiki/view/SO/SO1213AL/SistemiOperativi12CFUModulo220202021#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 indicata `/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.1.20202021.tgz` e copiarlo in una directory. All'interno di tale directory, dare il seguente comando:

```
tar xfzp so2.grader.1.20202021.tgz && cd grader.1
```

È ora necessario copiare il file `so2.2020.2021.1.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 `so2.2020.2021.1.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.

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`):

- -l
- -h
- -s regex (default: vuoto; nel seguito, sia s il valore dato a tale opzione).
- -S string (default: vuoto; nel seguito, sia S il valore dato a tale opzione).
- -e hexcont (default: vuoto; nel seguito, sia e il valore dato a tale opzione).

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] [-s string] [-S string] [-e string] [dirs]`, con p nome dello script, e lo script dovrà terminare con exit status 100. Nel caso 3, occorre precedere tale scritta con la seguente (sempre su standard error): `Non e' possibile dare contemporaneamente -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.

- 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 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.

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 si deve comportare in modo molto simile al comando `du -sb`. Pertanto, deve scrivere su standard output, per ogni directory d_i (escluse quelle da ignorare come descritto sopra), il nome di d_i (come dato per argomento) seguito dal numero totale di bytes necessari per memorizzare tutti i file e tutte le directory nell'albero radicato in d_i. Eventuali sottodirectory di d_i che non abbiano i permessi di scrittura ed esecuzione nei gruppi *group* e *other* (come sopra) vanno ignorate, con tutti loro files e sottodirectory, senza scrivere nulla su alcun file descriptor. Qualora ci siano dei file che siano hard link ad un altro file contenuto sempre nello stesso albero radicato in d_i, vanno contati una volta sola se `-h` è stato dato, altrimenti vanno contati tutti separatamente. Qualora ci siano dei file che siano soft link ad un altro file, occorre contare la loro dimensione effettiva se `-l` non è stato dato, altrimenti va contata la dimensione del file puntato (qualora ci sia una catena di soft links, considerare solo la destinazione finale). Qualora s non sia vuoto, è necessario considerare solo file o directory il cui nome (solo il nome, senza contare il path relativo) faccia match con l'espressione regolare s. Qualora S non sia vuoto, è necessario considerare solo i file (dentro la directory d_i) il cui testo contenga una sottostringa che faccia match con la stringa S. Qualora e non sia vuoto, è necessario considerare solo i file (dentro la directory d_i) il cui dump esadecimale contenga una sottostringa che faccia match con l'espressione regolare e. Se la dimensione D del file non è un multiplo di 4 bytes, riempire gli ultimi D mod 4 bytes con zeri. Infine, deve scrivere, sempre su standard output, su una sola riga `Total` T, dove T è la somma delle dimensioni scritte in precedenza. Se l'argomento `-h` è stato dato, occorre escludere dal conto gli eventuali hard link tra le diverse directory date come argomento. Pertanto riassumendo: se un file è hard linked $k_1, \ldots, k_n$ volte all'interno delle directory $d_1, \ldots, d_n$, allora, con l'opzione `-h`, va contato 1 volta per ogni d_i, e poi 1 volta complessivamente per il totale; altrimenti, senza l'opzione `-h`, va contato k_i volte per ogni d_i e poi $\sum_{i=1}^{n} k_i$ volte per il totale.

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.

Attenzione: non è permesso usare Python, Java, Perl o GCC, e nemmeno il programma `du`. 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. lista di comandi C da lanciare con rispettivi argomenti, terminati da un doppio carattere *underscore* (__); l'ultimo comando va terminato con un triplo carattere underscore (___);
3. lista di nomi di file.

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

Lo script dovrà lanciare in background e monitorare i comandi in C, ridirigendo sia lo standard output che lo standard error. A tal proposito, la lista di nomi di file data come ultimo argomento dovrà essere del tipo $f_1, f_2, \ldots, f_n, g_1, \ldots, g_n$, dove n è il numero di comandi in C. Sia $C = C_1, \ldots, C_n$; allora lo standard output del comando C_i va rediretto in f_i e lo standard error in g_i. Una mancata concordanza tra numero di comandi e numero di file dovrà portare lo script a terminare con exit status 130, visualizzando su standard error `Uso: s sampling commands files` (dove s è il nome dello script). Qualora uno dei comandi in C 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:

$$p_i(t_{i1} \ldots t_{ik_i})p_{i_j}$$

dove la riga i-esima ha le seguenti proprietà:

- p_i, p_{i_j} sono PID di processi risultanti dall'esecuzione dei comandi in C; p_{i_j} è figlio di p_i;
- $t_{i1} \ldots t_{ik_i}$ sono thread all'interno del processo con PID p_i (che ne è il tgid).

Le radici della foresta p_i devono corrispondere a tutti e soli i processi di C. Le righe vanno ordinate (numericamente) prima sul primo PID, e poi sull'ultimo (ovviamente, a parità di primo PID, anche quelli tra parentesi saranno uguali). Notare che, a parità di primo PID, la lista di thread tra parentesi sarà la stessa.

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 in C).

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 problema MILP (Mixed Integer Linear Programming) richiede di ottimizzare, trovandone il minimo od il massimo, una funzione lineare definita su un insieme di variabili X, rispettando vincoli lineari di uguaglianza e/o disuguaglianza, definiti su un insieme di variabili Y (tipicamente, $X \subseteq Y$). Tali vincoli sono tipicamente divisi tra vincoli che coinvolgono almeno 2 variabili e vincoli che ne coinvolgono una sola (*bounds*). Le variabili X possono avere valori nei reali, negli interi o addirittura nei booleani.

Uno dei formati più usati per rappresentare un MILP problem è il seguente (con qualche semplificazione):

d
`obj:` $\sum_{x \in X} c_x x$
`Subject To`
l_1`:` $\sum_{x \in Y_1} c_{1x} x \leq b_1$
$\vdots$
l_n`:` $\sum_{x \in Y_n} c_{nx} x \leq b_n$
`Bounds`
$b_{11} \leq x_1 \leq b_{21}$
$\vdots$
$b_{1|X \cup Y|} \leq x_{|X \cup Y|} \leq b_{2|X \cup Y|}$
`Integer`
$x_{i_1} \dots x_{i_k}$
`Binary`
$x_{j_1} \dots x_{j_m}$
`End`

dove valgono le seguenti proprietà:

- d può essere o `Maximize` o `Minimize`
- $Y = \cup_{i=1}^{n} Y_i$
- $x_{i_1}, \dots, x_{i_k}$ sono le variabili intere; la lista può essere separata da spazi ed andate a capo (anche mischiati)
- $x_{j_1}, \dots, x_{j_m}$ sono le variabili booleane; la lista può essere separata da spazi ed andate a capo (anche mischiati)
- $l_1, \dots, l_n$ sono identificatori, con $l_i \neq l_j$ per $i \neq j$; possono essere anche vuoti (nel qual caso, non ci sono neanche i 2 punti)
- $b_1, \dots, b_n, b_{11}, \dots, b_{2|X \cup Y|}$ sono valori reali, che possono essere rappresentati sia in virgola fissa che mobile (stessa sintassi di C, Java e Python)

- per ogni $x \in X \cup Y$, $c_x, c_{1x} \ldots, c_{nx}$ sono anch'essi valori reali, che possono essere rappresentati sia in virgola fissa che mobile (stessa sintassi di C, Java e Python), e devono essere separati da almeno uno spazio rispetto alla variabile x che moltiplicano. Se un c_x o un c_{ix} manca, va inteso come avente valore 1 (con segno `+`) o -1 (con segno `-`).

Si richiede quindi di scrivere uno script `gawk`, da chiamare `3.awk`, che prenda in input un file di configurazione ed 2 file formattati come descritto sopra; nel seguito, indicheremo tali file con I, F_1, F_2 (da considerare nell'ordine con cui vengono dati in input). Il file I è formattato come segue:

`intersection=`i
`union=`u
`differences=`d
`res=`r

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. Siano $i, u, d \in \{0, 1\}$ i valori che risultano assegnati con queste regole a `intersection`, `union` e `differences` (nel caso non siano presenti, valgono 0). Per quanto riguarda il valore r assegnato a `res`, può essere una qualsiasi stringa (vuoto se `res` non è presente in I).

Lo script `3.awk` dovrà quindi 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 stampare, scrivendolo sul file r (se r è vuoto, allora occorre scriverlo su standard output), nell'ordine:

- solo se $i = 1$, una riga `Intersection:` a, seguita da un'altra riga che inizi con un TAB e seguita dai nomi (sempre su questun'unica riga, separati da spazi) delle variabili che si trovano in entrambi i problemi MILP F_1, F_2 (dove a è il numero di tali variabili):

- solo se $u = 1$, una riga `Union:` a, seguita da un'altra riga che inizi con un TAB e seguita dai nomi (sempre su questun'unica riga, separati da spazi) delle variabili che si trovano in almeno uno dei problemi MILP F_1, F_2 (dove a è il numero di tali variabili);

- solo se $d = 1$, una riga `Differences:` a, con $a = b + c + d$, seguita da 4 righe:

  - la prima deve iniziare con un TAB, seguito dalla scritta `coeffs` b: a sua volta seguita dalle variabili che compaiono con coefficienti diversi, con formato x c_{1x} c_{2x}, dove x è una variabile che compare nell'obiettivo di F_1 con coefficiente c_{1x} e con coefficiente c_{2x} nell'obiettivo di F_2, e si ha $c_{1x} \neq c_{2x}$ (scrivere sempre il segno attaccato al coefficiente); infine, b è il numero di tali variabili;

– la seconda deve iniziare con un TAB, seguito dalla scritta `same coeffs` b'`:` a sua volta seguita dai nomi delle variabili, con formato x c_x (scrivere sempre il segno attaccato al coefficiente), che compaiono con lo stesso coefficiente c_x sia nell'obiettivo di F_1 che in quello di F_2, dove b' è il numero di tali variabili;

– la terza deve iniziare con un TAB, seguito dalla scritta `only in` F_1 c`:` a sua volta seguita dai nomi delle variabili che compaiono solo nell'obiettivo di F_1 e non in quello di F_2, dove c è il numero di tali variabili;

– la quarta deve iniziare con un TAB, seguito dalla scritta `only in` F_2 d`:` a sua volta seguita dai nomi delle variabili che compaiono solo nell'obiettivo di F_2 e non in quello di F_1, dove d è il numero di tali variabili.

In tutti i casi, occorre che ogni riga sia ordinata lessicograficamente sul nome delle variabili.

Lo script non deve scrivere nulla sullo standard error, tranne che nei seguenti casi:

- nella prima riga, per riportare gli argomenti come descritto sopra;
- non vengono dati esattamente 3 file di input; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: dare 3 file di input`, e terminare senza effettuare altre computazioni con exit status 10;
- nel file di configurazione, il valore di r contiene un path assoluto; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: non e' possibile specificare path assoluti`, e terminare senza effettuare altre computazioni con exit status 20.

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`.