Sistemi Operativi, Secondo Modulo
A.A. 2020/2021
Testo del Secondo Homework

Igor Melatti

Come si consegna

Il presente documento descrive le specifiche per l'homework 2. Esso consiste di 3 esercizi, per risolvere i quali occorre creare 3 cartelle, con la cartella di nome i che contiene la soluzione all'esercizio i. La directory `1` dovrà contenere almeno un file `1.c` ed un file `Makefile`. Quest'ultimo dovrà generare un file `1` quando viene invocato. La directory `2` dovrà contenere almeno un file `2.server.c`, un file `2.client.c` ed un file `Makefile`. Quest'ultimo dovrà generare un file `2.server` ed un file `2.client` quando viene invocato. Infine, la directory `3` dovrà contenere almeno un file `3.c` ed un file `Makefile`. Quest'ultimo dovrà generare un file `3` quando viene invocato. Tutti i `Makefile` devono anche avere un'azione `clean` che cancella i rispettivi file eseguibili. Per consegnare la soluzione, seguire i seguenti passi:

1. creare una directory chiamata `so2.2020.2021.2.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare le directory `1`, `2` e `3` in `so2.2020.2021.2.matricola`
3. creare il file da sottomettere con il seguente comando: `tar cfz so2.2020.2021.2.matricola.tgz [1-3]`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=120` e uploadare il file `so2.2020.2021.2.matricola.tgz` ottenuto al passo precedente.

Come si auto-valuta

Per poter autovalutare il proprio homework, occorre 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, e dal prossimo riavvio di VirtualBox sarà possibile accedere alla cartella x del sistema operativo ospitante tramite la cartella `/media/sf_`y di Debian. Infine, è necessario installare il programma `bc` dando il seguente comando da terminale: `sudo apt-get install bc`.

All'interno delle suddette macchine virtuali, scaricare il pacchetto per l'autovalutazione (*grader*) dall'URL `151.100.17.205/download_from_here/so2.grader.2.20202021.tgz` e copiarlo in una directory con permessi di scrittura per l'utente attuale. All'interno di tale directory, dare il seguente comando:

```
tar xfzp so2.grader.2.20202021.tgz && cd grader.2
```

È ora necessario copiare il file `so2.2020.2021.2.matricola.tgz` descritto sopra dentro alla directory attuale (ovvero, `grader.2`). Dopodiché, è sufficiente lanciare `grader.2.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.

Nota bene bis: tutti i test vengono avviati con `valgrind`, per controllare che non ci siano errori nella gestione della memoria.

Esercizio 1

Scrivere un programma C che implementi esattamente (tranne una piccola differenza negli output sui file descriptor 3, 4 e 5, vedere più sotto) la stessa applicazione del primo esercizio del primo homework. Pertanto, dovrà avere la seguente sinossi:

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

dove le opzioni sono le seguenti (si consiglia l'uso della funzione standard `getopt`):

- -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 dal comando 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 del comando, e il comando 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 al comando (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.

Il comando 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, il comando 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 del comando deve essere il numero di file o directory, tra gli argomenti del comando, 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. Nel realizzare tale specifica, considerare l'ordinamento standard sui caratteri (usato, ad esempio, da `strcmp`), basato sul corrispondente codice ASCII (ad esempio, il carattere slash `/` è più piccolo della lettera `f`, in quanto i rispettivi codici ASCII sono 47 e 102).

Attenzione: non è permesso usare le system call `system`, `exec`, `sleep` e `popen`. Per la verifica del matching con un'espressione regolare, vedere `man 3 regex`. Assumere che i file descriptor 3, 4 e 5 siano già aperti al momento dell'invocazione del comando. Il programma 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. Per ogni test definito nella valutazione, il programma dovrà ritornare la soluzione dopo al più 10 minuti.

Esempi

Da dentro la directory `grader.2`, dare il comando `tar xfzp all.tgz input_output.1 && cd input_output.1`. Ci sono 6 esempi di come il programma `./1/1` possa 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. La directory con l'output atteso è `check/out.`i. La directory `check/out_tmp.`i contiene dei log di esempio di `valgrind`. Quelli prodotti dalla soluzione proposta dovranno essere simili a questi, ovvero non contenere messaggi d'errore (questo controllo viene fatto in `inp_out.`i`.sh`).

Esercizio 2

Scrivere due programmi C, un client (`2.client.c`) ed un server (`2.server.c`), che comunichino tramite named pipes. Più in dettaglio, il client dovrà avere i seguenti argomenti (nell'ordine dato):

- un'opzione `-f` con argomento f (se non data, f è vuoto);
- due nomi di named pipes n_r, n_w;
- due nomi di file g_1, g_2;
- eventuali altri argomenti a.

Il server, invece, dovrà avere due argomenti: due nomi di named pipe n_r, n_w.

Il server dovrà creare le named pipe, se non esistono. Dopodiché, il server deve leggere dalla pipe n_r, interpretare ciò che legge come testo, ed ogni riga di tale testo va intesa come una richiesta, che deve essere servita eseguendo il programma `/bin/bash`. Più in dettaglio, l'input letto dalla named pipe va passato allo standard input del programma `/bin/bash`, con ambiente aumentato da a (ovvero, quanto passato come opzione ulteriore al client; assumere che a, se dato, sia una lista di assegnamenti separati da spazi). La risposta di `/bin/bash`, che sia su standard output od error, va rimandata al client tramite la named pipe n_w, e scritta sullo standard output del server stesso (in caso ci siano sia standard output che standard error, scrivere prima tutto lo standard output e poi tutto lo standard error).

Il server potrà essere terminato se un client manda su n_r una riga il cui unico contenuto sia `EXIT`. In tal caso, tutti i figli eventuamente creati dal server dovranno essere terminati; eventuali altre righe dopo `EXIT` vanno ignorate.

Il client, invece, dovrà mandare al server, tramite n_w, tutto quanto letto dal file f, o da standard input se quest'ultimo è vuoto. Ogni riga ricevuta dal server (su n_r) va scritta su g_1 se era nello standard output della risposta di `/bin/bash`, e su g_2 altrimenti.

Si possono manifestare solamente i seguenti errori:

- Il server non viene avviato con i 2 argomenti richiesti. Il programma dovrà allora terminare con exit status 10 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `piperd pipewr`, dove p è il nome del programma stesso.
- Una delle pipe n_r, n_w passate al server non esiste, ma non è possibile crearla. Il programma dovrà allora terminare con exit status 40 (senza eseguire alcuna azione), e scrivendo su standard error `Unable to create named pipe` n `because of` e, dove e è la stringa di sistema che spiega l'errore e n il nome della pipe che ha causato l'errore.
- Una delle pipe n_r, n_w passate al client non esiste. Il programma dovrà allora terminare con exit status 80 (senza eseguire alcuna azione),

e scrivendo su standard error `Unable to open named pipe` n `because of` e, dove e è la stringa di sistema che spiega l'errore e n il nome della pipe che ha causato l'errore.

- Uno degli argomenti n_r, n_w passati al server o al client esiste, ma non è una pipe. Il programma dovrà allora terminare con exit status 30 (senza eseguire alcuna azione), e scrivendo su standard error `Named pipe` n `is not a named pipe`, dove n è il nome della pipe che ha causato l'errore.

- Il client non viene avviato con almeno 4 argomenti. Il programma dovrà allora terminare con exit status 20 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `[opts] piperd pipewr file1 file2 [assigns]`, dove p è il nome del programma stesso.

- Fallisce una qualsiasi altra system call. Il programma dovrà allora terminare con exit status 100 (senza eseguire alcun'altra azione), e scrivendo su standard error `System call` s `failed because of` e, dove e è la stringa di sistema che spiega l'errore ed s è la system call che ha fallito.

Attenzione: non è permesso usare le system call `system`, `popen` e `sleep`. I programmi non devono scrivere nulla sullo standard error, a meno che non si tratti di un errore nelle opzioni da riga di comando come descritto sopra. Per ogni test definito nella valutazione, i programmi dovranno ritornare la soluzione dopo al più 10 minuti.

Suggerimento: ovviamente, il client deve inviare al server il suo argomento a (se dato), di modo che il server lo possa passare a `/bin/bash`. Per distinguere a dal resto dell'input, conviene usare un mini-protocollo: prima il client invia la lunghezza di a, poi a stesso. In tal modo, il server sa quando ha finito di leggere a e può quindi avviare `/bin/bash`, passandogli come opzione ciò che legge dalla pipe.

Suggerimento bis: attenzione ad evitare stalli all'avvio di server e client; anche qui un mini-protocollo può aiutare.

Esempi

Da dentro la directory `grader.2`, dare il comando `tar xfz all.tgz input_output.2 && cd input_output.2`. Ci sono 6 esempi di come i programmi `./2/2.server` e `./2/2.client` possano essere lanciati, 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. La directory con l'output atteso è `check/out.`i. La directory `check/out_tmp.`i contiene dei log di esempio di `valgrind`. Quelli prodotti dalla soluzione proposta dovranno essere simili a questi, ovvero non contenere messaggi d'errore (questo controllo viene fatto in `inp_out.`i`.sh`).

Esercizio 3

Scrivere un programma C di nome `3.c`, in grado di leggere e modificare file binari con un certo formato. Più in dettaglio, il programma dovrà prendere almeno 6 argomenti: il nome di un file f_{in}, il nome di un file f_{out}, due numeri positivi n, c (con $c \leq 8$) ed almeno 2 altri argomenti (nel seguito, ci riferiremo a questi ulteriori argomenti, che devono essere almeno 2, con s). Il file f_{in} è un file binario di un testo "offuscato", organizzato come segue: una intestazione costituita da un intero positivo a 4 byte i e da un intero su 3 bit b, seguita dal contenuto vero e proprio del file, che conterrà almeno i bytes. I bytes del contenuto sono organizzati come segue: i primi i sono codici ASCII in cui i b bit più significativi sono scambiati con gli $8-b$ bit meno significativi (ad es.: se $b = 3$ e un byte vale 11000101, allora il byte risultante sarà 00101110), mentre i bytes che vanno da $i+1$ in poi sono codici ASCII standard. Gli ultimi $8-b$ bit del file sono riempiti con altrettanti 0. Pertanto, per ricomporre il testo non offuscato, occorre leggere l'intestazione per avere i e b, per poi leggere i primi i bytes del contenuto scambiando nuovamente i bit, e poi leggere normalmente i bytes restanti, prestando attenzione al fatto che ogni byte del contenuto è "spezzato" in 2 byte del file binario.

Il programma dovrà eseguire il comando `/usr/bin/tr` s, facendo in modo che lo standard input di tale comando sia costituito dal contenuto di f_{in} "de-offuscato" (ovvero, f_{in} va assunto essere un file offuscato come descritto sopra, ed occorre ricostruirne il testo come descritto sopra). Il programma dovrà poi prendere il risultato del comando sopraindicato, "rioffuscarlo" e scrivere il risultato di tale "rioffuscamento" su f_{out}. Il rioffuscamento va effettuato usando n, c come nuovi i, b. Il risultato è da intendersi come quanto viene scritto da `/usr/bin/tr` su standard output, ignorando eventuali scritte su standard error.

Si possono manifestare solamente i seguenti errori:

- Il programma `3` non viene avviato con gli argomenti richiesti. Il programma dovrà allora terminare con exit status 10 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `filein fileout i b tr_arg1 tr_arg2 [tr_other_args]`, dove p è il nome del programma stesso.
- Il file f_{in} non esiste o non è accessibile in lettura. Il programma dovrà allora terminare con exit status 20 (senza eseguire alcuna azione), e scrivendo su standard error `Unable to open file` f_{in} `because of` e, dove e è la stringa di sistema che spiega l'errore.
- Il file f_{in} letto da `3` non è ben formattato: non contiene almeno 5 bytes, oppure non contiene almeno $n+5$ bytes. In questo caso, `3` deve terminare con exit status 30, generando un file di output di dimensione 0 e scrivendo su standard error `Wrong format for input binary file` f_{in}.
- La risposta di `/usr/bin/tr` non ha almeno i bytes. Allora, il file andrà creato con $i = 0$ (quindi, senza offuscamento ma con i primi 5 bytes speciali), e `3` devrà terminare con exit status 80.

- Il file f_{out} non può essere creato o sovrascritto. Il programma dovrà allora terminare con exit status 70 (senza eseguire alcuna azione), e scrivendo su standard error `Unable to open file` f_{out} `because of` e, dove e è la stringa di sistema che spiega l'errore.

- Fallisce una qualsiasi altra system call. Il programma dovrà allora terminare con exit status 100 (senza eseguire alcun'altra azione), e scrivendo su standard error `System call` s `failed because of` e, dove e è la stringa di sistema che spiega l'errore ed s è la system call che ha fallito.

Attenzione: non è permesso usare le system call `system`, `popen` e `sleep`. Il programma 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. Per ogni test definito nella valutazione, il programma dovrà ritornare la soluzione dopo al più 10 minuti.

Esempi

Da dentro la directory `grader.2`, dare il comando `tar xfz all.tgz input_output.3 && cd input_output.3`. Ci sono 6 esempi di come il programma `./3/3` possa essere lanciato, salvati in file con nomi `inp_out.`i`.sh` (con $i \in \{1, \ldots, 6\}$). Per ciascuno di questi script, i file di input sono nella directory `inp.`i. La directory con l'output atteso è `check/out.`i. La directory `check/out_tmp.`i contiene dei log di esempio di `valgrind`. Quelli prodotti dalla soluzione proposta dovranno essere simili a questi, ovvero non contenere messaggi d'errore (questo controllo viene fatto in `inp_out.`i`.sh`).