Sistemi Operativi, Secondo Modulo
A.A. 2019/2020
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 `so2.2019.2020.2.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare le directory `1`, `2` e `3` in `so2.2019.2020.2.matricola`
3. creare il file da sottomettere con il seguente comando: `tar cfz so2.2019.2020.2.matricola.tgz [1-3]`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=94` e uploadare il file `so2.2019.2020.2.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/`

`SistemiOperativi12CFUModulo220192020#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.2.20192020.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.2.20192020.tgz && cd grader.2
```

È ora necessario copiare il file `so2.2019.2020.2.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 `so2.2019.2020.1.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 un `find` semplificato (nel seguito, quando si parla di "file" si intende un file o una directory). In particolare, il programma potrà essere lanciato con i seguenti argomenti:

1. lista di file (almeno uno), separati da spazi (assumere che nessuno di questi file possa cominciare con un dash `-`);
2. argomento opzionale `-maxdepth` N, con lo stesso significato che ha nel `find` di sistema (ovvero: un file viene considerato solo se il suo attuale path, relativo alla directory data come argomento, contiene al più N directory);
3. lista di test, separati da spazi;
4. lista di azioni, separate da spazi.

Dei test, vanno implementati solo i seguenti:

- `-type` t, con lo stesso significato che ha nel `find` di sistema, ma con t che può essere solo uno tra `l`, `d` o `f`;
- `-size` N, con lo stesso significato che ha nel `find` di sistema, ma senza suffissi ed assumendo che N indichi sempre i bytes (quindi, come il suffisso `c` di sistema);
- `-perma` i, con lo stesso significato che ha nel `find` di sistema, ma i è sempre fatto di sole 4 cifre ottali (quindi, niente permessi definiti in modo simbolico o caratteri speciali come `-`).

Quando più test sono presenti, assumere di dover effettuare l'OR degli stessi. Se sono presenti più istanze dello stesso test, considerare solo l'ultima istanza. Se non c'è nessun test, allora tutti i file passano la fase di test. Non è possibile effettuare altre combinazioni logiche dei test.

Delle azioni, vanno implementate solo le seguenti:

- `-ls`, con un significato simile a quello che ha nel `find` di sistema: se il file matchato ha path `file`, dovrà restituire una selezione dell'output del comando `ls -ld --color=never file`: vanno stampati solo i permessi, l'hard link count, la dimensione ed il nome; come separatore tra questi campi, usare sempre il tab. Se il file è un link simbolico, va mostrata la destinazione, come con l'opzione `-l` di `ls`;
- `-print`, con lo stesso significato che ha nel `find` di sistema.

Se non c'è nessuna azione, effettuare l'azione `-print`. Se c'è più di una azione, effettuare l'ultima azione data.

Si possono manifestare solamente i seguenti errori:

- Non esiste una delle directory D date nella lista iniziale dei file. Il programma dovrà allora proseguire con la prossima directory, scrivendo su standard error `Unable to open` D `because of` e, dove e è la stringa di sistema che spiega l'errore. L'exit status del programma dovrà essere il numero di directory saltate per via di questo motivo.

- Non viene dato nessun file nella lista dei file. Il programma dovrà allora terminare con exit status 20 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `dirs [-maxdepth` N`] [tests] [actions]`, 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`, `exec`, `sleep` e `popen`. 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):

- due nomi di named pipe n_r, n_w;
- un intero f;
- 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 `/usr/bin/bc`. Più in dettaglio, l'input letto dalla named pipe va passato allo standard input del programma `/usr/bin/bc` con opzioni a (ovvero, quelle passate al client dopo il terzo argomento). Il programma `bc` si comporta nel seguente modo: per ogni riga ricevuta in input, scrive una risposta su standard output o su standard error, su una o più righe. Tale risposta, se su standard output, va rimandata al client tramite la named pipe n_w, senza alcuna modifica. Se invece la risposta di `bc` è su standard error, occorre prima di tutto eliminare, da ogni riga della risposta stessa, la parte iniziale, ovvero dal primo carattere fino al carattere che segue il primo `:`. Ad esempio, se la risposta consiste nelle due righe:

```
(standard_in) 1: illegal character: G
(standard_in) 2: illegal character: H
```

allora occorre ottenere le righe:

```
illegal character: G
illegal character: H
```

A questo punto, il risultato va mandato sia sulla named pipe n_w al client, sia sullo standard output del server stesso.

Il server potrà essere terminato se un client manda il messaggio `EXIT` seguito da andata a capo. In tal caso, tutti i figli eventuamente creati dal server dovranno essere terminati.

Il client, invece, dovrà mandare al server, tramite n_w, tutto quanto letto dal file descriptor f (da assumere come già aperto). Ogni riga ricevuta dal server (su n_r) va scritta sullo standard output del client stesso.

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 3 argomenti. Il programma dovrà allora terminare con exit status 20 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `piperd pipewr fd [opts]`, 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 `bc`. 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 `bc`, 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, `3.c`, in grado di scrivere e di interpretare file binari con un certo formato. Più in dettaglio, il programma dovrà prendere come argomento un nome di un file f ed un operatore w, più un argomento opzionale i tra 1 e 9.

Il file f è un file binario costituito da blocchi contigui così formati:

```
selettore
ripetizioni
struttura
```

Il campo `selettore` è un valore intero tra 1 e 3, memorizzato in 1 byte, mentre il campo `ripetizioni` è un valore intero tra 0 e 255, sempre memorizzato in un byte. Attenzione: tra selettore, ripetizioni e struttura non ci sono byte di separazione. A seconda del valore del selettore, la struttura è così formata:

1. un double, un float, e 7 char;
2. 2 interi senza segno, 2 interi con segno ed un float;
3. 50 interi con segno.

Ogni struttura è ripetuta tante volte quanto è il valore di `ripetizioni`. Non ci sono padding.

Il programma `3.c`, se l'argomento w vale `r`, deve leggere il file f e scriverlo come testo su standard output. Se invece l'argomento w ha qualsiasi altro valore, deve leggere i blocchi come testo da standard input e scriverli in binario sul file f. Per la lettura/scrittura di selettore, ripetizioni e struttura *come testo*, valgono le seguenti regole:

1. il selettore è separato dalle ripetizioni da uno spazio, ed entrambi sono separati dalla/e struttura/e corrispondente/i da una andata a capo;
2. ogni singolo valore di una struttura è separato dagli altri tramite uno spazio (i char sono sempre stampabili ma diversi dallo spazio);
3. i double sono in virgola fissa, i float in virgola mobile;
4. sia i float che i double hanno i cifre dopo la virgola;
5. gli interi con segno vanno sempre stampati con il segno, gli interi senza segno sempre senza segno;
6. al termine di ogni struttura c'è un a capo;
7. al termine di tutte le ripetizioni di ogni struttura ci sono 2 a capo.

Si possono manifestare solamente i seguenti errori:

- Il programma non viene avviato con meno di 2 o più di 4 argomenti. Il programma dovrà allora terminare con exit status 10 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `file mod [digits]`, dove p è il nome del programma stesso.

- Il programma viene avviato con opzione w diversa da `r`, e con 3 argomenti complessivi. Il programma dovrà allora terminare con exit status 20 (senza eseguire alcuna azione), e scrivendo su standard error `If mod is not r, then only the input file must be given`.

- Il programma viene avviato con opzione w uguale a `r`, e con 2 argomenti complessivi. Il programma dovrà allora terminare con exit status 30 (senza eseguire alcuna azione), e scrivendo su standard error `If mod is r, then also the number of digits for float and double must be given`.

- Il file f non esiste o non è accessibile. Il programma dovrà allora terminare con exit status 40 (senza eseguire alcuna azione), e scrivendo su standard error `Unable to` r `file` f `because of` e, dove e è la stringa di sistema che spiega l'errore e r è `read from` se w è `r` e `write to` altrimenti.

- Il file f letto dal programma con opzione w uguale ad `r` non è ben formattato: c'è un selettore con valore non compreso tra 1 e 3, oppure alla fine del file non ci sono abbastanza dati per l'intera struttura "promessa" dal selettore. In tale caso, il programma deve effettuare la traduzione fino all'errore escluso, e terminare con exit status 50.

- 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`).