Sistemi Operativi, Secondo Modulo
A.A. 2021/2022
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 nominate `1`, `2` e `3`, dove la cartella i contiene la soluzione all'esercizio i. Per le directory `1` e `3`, l'esecuzione del comando `make` dovrà creare un file eseguibile `1` e `3`, rispettivamente, che risolvano l'esercizio corrispondente. Per la directory `2`, dovranno essere generait 2 file: `2.server` e `2.client`. Su ogni directory i deve anche essere possibile eseguire il comando `make clean`, che cancelli i rispettivi file eseguibili. Per consegnare la soluzione, seguire i seguenti passi:

1. creare una directory chiamata `so2.2021.2022.2.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare le directory `1`, `2` e `3` in `so2.2021.2022.2.matricola`
3. creare il file da sottomettere con il seguente comando: `tar cfz so2.2021.2022.2.matricola.tgz [1-3]`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=159` e uploadare il file `so2.2021.2022.2.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/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, 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.2.20212022.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.2.20212022.tgz && cd grader.2
```

È ora necessario copiare il file `so2.2021.2022.2.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 `so2.2021.2022.1.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.

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 di nome `1.c` che implementi un `tree` semplificato. In particolare, il programma dovrà avere la seguente sinossi:

`1 [options] [directories]`

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

- `-P` *p*
- `-a`
- `-p`

Se invocato con la riga di comando `./1` *opts dirs*, il programma deve restituire lo stesso output del comando `LC_ALL=C tree -n --charset=ascii` *opts dirs*.

Si possono manifestare solamente i seguenti errori:

- Una delle directory date come argomento è in realtà un file. Il programma dovrà allora scrivere a fianco del nome del file `[error opening dir because of being not a dir]`; al termine dell'esecuzione, l'exit status dovrà essere 10.
- Viene data un'opzione non corretta. Il programma dovrà allora terminare con exit status 20 (senza scrivere nessun output), e scrivendo su standard error `Usage:` *p* `[-P pattern] [-ap] [dirs]`, dove *p* è il nome del programma stesso.
- Fallisce una qualsiasi altra system call. Il programma dovrà allora terminare con exit status 100 (senza scrivere nessun altro output), 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.

In tutti gli altri casi, l'exit status dev'essere 0.

Attenzione: non è permesso usare le system call `system`, `exec`, `popen` e `sleep`. Inoltre, l'ambiente in cui verrà eseguito il grader non dispone di un comando `tree`. Il programma non deve scrivere nulla sullo standard error, a meno che non si tratti di uno degli errori descritti sopra. Per ogni test definito nella valutazione, il programma dovrà ritornare la soluzione dopo al più 10 minuti.

Suggerimento: usare le funzioni `fnmatch` e `scandir`.

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 pipes c_r, c_w;
- una stringa p;
- eventuali altri argomenti $a_1, \ldots, a_k$.

Il server, invece, dovrà avere due argomenti: due nomi di named pipe s_w, s_r.

Il server dovrà creare le named pipe, se non esistono. Quando arriva un nuova richiesta su s_r, la deve servire eseguendo il file p con argomenti $a_1, \ldots, a_k$. Sullo standard input di p, il server deve mandare tutto quanto viene letto dal client su standard input, effettuando un'invocazione di p per ogni singola riga letta. Il programma p scriverà una risposta su standard output e/o su standard error, su una o più righe. Tale risposta va rimandata al client usando s_w. Inoltre, va anche stampata sullo standard output del server: prima la risposta originariamente su standard output, poi quella su standard error. È possibile assumere che le risposte di p contengano solo caratteri ASCII standard, e che alla fine della risposta di p ci sia una andata a capo (sia su standard output che su standard error).

Il server potrà essere terminato se riceve dal client, nello stream di input da passare a p, una riga contenente solo `EXIT`.

In tal caso, tutti i figli eventuamente creati dal server dovranno essere terminati.

Il client, invece, dovrà mandare al server (tramite c_w) tutto quanto letto dallo standard input. Se legge `EXIT` su una singola riga, deve inviare la riga e poi terminare immediatamente. Ogni riga ricevuta dal server (tramite c_r) va scritta sul corrispondente stream del client: su standard output se il server l'aveva ricevuta dallo standard output di p, e sullo standard error 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 s_r, s_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 c_r, c_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 file [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 uno dei casi esplicitamente menzionati sopra (errori nelle opzioni). Per ogni test definito nella valutazione, i programmi dovranno ritornare la soluzione dopo al più 10 minuti.

Attenzione 2: ci sono alcune esecuzioni dei programmi p che stampano degli errori standard (ad esempio, per file che non esistono). Nella macchina virtuale, questi errori sono scritti in italiano, ma nella soluzione desiderata devono essere in inglese. Per ottenere questo effetto, è sufficiente che le esecuzioni dei programmi p avvengano in un ambiente aumentato con `LANG=sc_IT.UTF-8`.

Suggerimento: come prima cosa, il client deve inviare al server il nome del file da eseguire, nonché le opzioni da passargli, se presenti. Per distinguere queste comunicazioni dal resto dell'input, conviene usare un mini-protocollo: ad esempio, mandando prima la lunghezza di p ed opzioni, e poi il messaggio stesso.

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 3 argomenti: il nome di un file f_{in}, il nome di un file f_{out} e una stringa s. Il file f_{in} è un file binario di un testo "offuscato", in cui, per ogni carattere memorizzato, viene usato un numero a 16 bit così composto: $\texttt{F}xy\texttt{F}_{16}$, dove xy_{16} è il codice ASCII del carattere. Il programma dovrà eseguire il comando `/bin/dd` s, facendo in modo che lo standard input di tale comando sia costituito dal contenuto del file "deoffuscato" (ovvero, togliendo i 4 bit a 1 prima e dopo ogni carattere). Il programma dovrà poi prendere il risultato del comando sopraindicato (su standard output), "rioffuscarlo" (aggiungendo 4 bit a 1 prima e dopo ogni carattere) e scrivere il risultato di tale "rioffuscamento" su f_{out}. Eventuali scritte di `dd` su standard error vanno ignorate.

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 `file_in file_out dd_args`, dove p è il nome del programma stesso.
- Il file f non esiste o non è accessibile. Il programma dovrà allora terminare con exit status 20 (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 f non è accessibile in lettura, oppure `write to` se non è accessibile in scrittura.
- Il file f_{in} letto da `3` non è ben formattato: mancano dei bit a 1 in alcune posizioni dove invece dovrebbero esserci. 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 `at byte` d, dove d è il numero del byte (a partire da 0) dove avviene 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.