Sistemi Operativi, Secondo Modulo
A.A. 2017/2018
Testo del Secondo Homework

Emanuele Gabrielli, 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` 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 `so2.2017.2018.2.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare le directory `1`, `2` e `3` in `so2.2017.2018.2.matricola`
3. creare il file da sottomettere con il seguente comando: `tar cfz so2.2017.2018.2.matricola.tgz [1-3]`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=44` e uploadare il file `so2.2017.2018.2.matricola.tgz` ottenuto al passo precedente.

Come si auto-valuta

Per poter autovalutare il proprio homework, è necessario installare VirtualBox (`https://www.virtualbox.org/`), creare una macchina virtuale da 32 bit ed indicare come disco di tale macchina virtuale quello corrispondente a Lubuntu 14.04-3, come scaricabile da `http://www.osboxes.org/lubuntu/`. È necessario installare `gawk`, in quanto in questa distribuzione di Lubuntu c'è invece mawk.

Per farlo, è sufficiente dare i seguenti comandi: `sudo apt-get update && sudo apt-get upgrade && sudo apt-get install gawk` (rispondere NO alla domanda sul passare alla versione successiva di Lubuntu). È inoltre necessario installare `valgrind`: basta dare il comando `sudo apt-get install valgrind`.

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 Lubuntu. Si consiglia inoltre di installare le "Guest Additions" nel seguente modo: dapprima dare il comando `sudo apt-get install dkms` da un terminale all'interno di Lubuntu, poi scegliere "Insert Guest Additions CD Image" dal menu di VirtualBox, e poi di nuovo da terminale di Lubuntu scrivere `cd /media/osboxes/VBOXADDITIONS*; sudo ./VBoxLinuxAdditions.run`. Infine, riavviare Lubuntu.

All'interno di tale macchina virtuale, scaricare il pacchetto per l'autovalutazione (*grader*) dall'URL `151.100.17.205/download_from_here/so2.grader.2.20172018.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 grader.2.tgz && cd grader.2
```

È ora necessario copiare il file `so2.2017.2018.2.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 `so2.2017.2018.2.matricola.tgz` contenga solo l'esercizio i).

Valutazione ed auto-valutazione: `valgrind`

Sia per la valutazione che per l'auto-valutazione, verrà usato il programma `valgrind`, per controllare che la memoria sia gestita in modo corretto (vedere sopra per l'installazione). In particolare, affinché valutazione ed auto-valutazione vadano a buon fine, è necessario che i messaggi di errore di valgrind *non* compaiano nell'output. Per una descrizione dei messaggi d'errore di `valgrind`, vedere `http://valgrind.org/docs/manual/mc-manual.html#mc-manual.errormsgs`.

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
- `-L` L
- `-a`

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*. Attenzione: i file che sono link simbolici non passano mai i test di `-P`.

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] [-L level] [-a] [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`, `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`.

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, all'interno della quale ci sono dei

file `res.`j`.`t`.txt`, contenente lo std$t$ della j-esima invocazione di `1` fatta dentro `inp_out.`i`.sh`. Infine, il file `res.make` contiene eventuali errori o warning in fase di compilazione (deve essere vuoto). 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`).

Attenzione: se viene lanciato il grader, la corrispondenza tra input ed output viene persa: ad `out.`i corrisponde un `check/out.`i' con $i \neq i'$. Fare riferimento a ciò che scrive il grader stesso per trovare la giusta corrispondenza input-output.

Esercizio 2

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

- il numero di una porta;
- una stringa s.

Il server, invece, dovrà avere un solo argomento: il numero di una porta.

Il server dovrà creare la socket e rimanere in ascolto sulla porta data; non appena sarà pronto ad accettare richieste di connessioni, dovrà scrivere "Ready to go!" sul file descriptor 3, da assumere come già aperto al momento di chiamare il server (tale scrittura dovrà essere seguita da un *flush* del file descriptor 3). Quando arriva un nuova richiesta, la deve servire eseguendo il programma `/usr/bin/awk`, cui dovrà passare, come programma da eseguire, la stringa s data input al client. L'input di `awk` sul server deve essere costituito da quanto viene letto dal client su standard input. Il programma `awk` scriverà una risposta su standard output e/o su standard error, su una o più righe. Tale risposta, se su standard output, va rimandata al client tramite la socket, senza alcuna modifica. Se invece la risposta di `awk` è su standard error, occorrerà prefiggere ogni riga con la scritta `ERROR`n`:`, dove n è il numero progressivo della riga di errore. Dopodiché, il risultato va mandato sia sul socket al client, sia sul file descriptor 4 (sempre da assumersi come già aperto) del server stesso. È possibile assumere che le risposte di `awk` contengano solo caratteri ASCII standard, e che alla fine della risposta di `awk` ci sia una andata a capo (sia su standard output che su standard error).

Il server potrà essere terminato se, come risultato dell'esecuzione di una richiesta di un client, ottiene come *exit status* di `awk` il valore 100. In tal caso, tutti i figli eventuamente creati dal server dovranno essere terminati.

Il client, invece, dovrà mandare al server tutto quanto letto dallo standard input. Ogni riga ricevuta dal server va scritta sul corrispondente stream del client: su standard output se il server l'aveva ricevuta dallo standard output di `awk`, e sullo standard error altrimenti.

Si possono manifestare solamente i seguenti errori:

- Il server non viene avviato con l'argomento richiesto. Il programma dovrà allora terminare con exit status 10 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `port_number`, dove p è il nome del programma stesso.
- Non è possibile usare numero di porta s passato al server (ad es. perché è già in uso). Il programma dovrà allora terminare con exit status 40 (senza eseguire alcuna azione), e scrivendo su standard error `Unable to create socket` s `because of` e, dove e è la stringa di sistema che spiega l'errore.

- Non è possibile accettare una richiesta proveniente da un client. Il server dovrà allora terminare con exit status 50 (senza eseguire alcun'altra azione), e scrivendo su standard error `Unable to accept incoming connection because of` e, dove e è la stringa di sistema che spiega l'errore.

- Il client non viene avviato con i 2 argomenti richiesti. Il programma dovrà allora terminare con exit status 20 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `port_number awk_program`, dove p è il nome del programma stesso.

- Il numero di porta s passato al client non esiste o non è accessibile. Il programma dovrà allora terminare con exit status 50 (senza eseguire alcuna azione), e scrivendo su standard error `Unable to read/write from socket` s `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`. 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.

Suggerimento: come prima cosa, il client deve inviare al server il suo secondo argomento s, di modo che il server lo possa passare ad `awk`. Per distinguere s dal resto dell'input, conviene usare un mini-protocollo: prima il client invia la lunghezza di s poi s stesso. In tal modo, il server sa quando ha finito di leggere s e avviare `awk`, passandogli come input ciò che legge dal socket.

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 $i \in \{1, \ldots, 6\}$). La directory con l'output atteso è `check/out.`i, all'interno della quale ci sono dei file `res.`j`.`t`.txt`, contenente, per $j \geq 1$ lo std$t$ della j-esima invocazione di `2.client` fatta dentro `inp_out.`i`.sh`. Invece, i file `res.0.*.txt` contengono l'output sul file descriptor t (per $t = 3, 4$) o su std$t$ (per $t \in \{err, out\}$) per l'unica invocazione di `2.server`. Il file `res.err.txt` deve essere vuoto, altrimenti contiene delle stringhe che catturano un qualche errore (ad esempio, ci sono ancora server in esecuzione quando non dovrebbero esserci). Infine, il file `res.make` contiene eventuali errori o warning in fase di compilazione (deve essere vuoto). 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`).

Attenzione: se viene lanciato il grader, la corrispondenza tra input ed output viene persa: ad `out.`i corrisponde un `check/out.`i' con $i \neq i'$. Fare riferimento a ciò che scrive il grader stesso per trovare la giusta corrispondenza input-output.

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: $0xy0_{16}$, dove xy_{16} è il codice ASCII del carattere. Il programma dovrà eseguire il comando `/bin/sed -e` s, facendo in modo che lo standard input di tale comando sia costituito dal contenuto del file "deoffuscato" (ovvero, togliendo i 4 bit a zero prima e dopo ogni carattere). Il programma dovrà poi prendere il risultato del comando sopraindicato (su standard output), "rioffuscarlo" (aggiungendo 4 bit a zero prima e dopo ogni carattere) e scrivere il risultato di tale "rioffuscamento" su f_{out}. Eventuali scritte di `sed` 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 sed_script`, 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 letto da `3` non è ben formattato: mancano dei bit a 0 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.

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 pro-

gramma `./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. Tale directory contiene il file di destinazione di dell'invocazione di `3` contenuta in `inp_out.`i`.sh`, nonché i file con il contenuto di stdout e stderr. Infine, il file `res.make` contiene eventuali errori o warning in fase di compilazione (deve essere vuoto). 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`).

Attenzione: se viene lanciato il grader, la corrispondenza tra input ed output viene persa: ad `out.`i corrisponde un `check/out.`i' con $i \neq i'$. Fare riferimento a ciò che scrive il grader stesso per trovare la giusta corrispondenza input-output.