Sistemi Operativi, Secondo Modulo
A.A. 2016/2017
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.read.c`, un file `3.write.c` ed un file `Makefile`. Quest'ultimo dovrà generare un file `3.read` ed un file `3.write` 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 `so2.2016.2017.2.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare le directory `1`, `2` e `3` in `so2.2016.2017.2.matricola`
3. creare il file da sottomettere con il seguente comando: `tar cfz so2.2016.2017.2.matricola.tgz [1-3]`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=21` e uploadare il file `so2.2016.2017.2.matricola.tgz` ottenuto al passo precedente. **Attenzione:** il suddetto link è raggiungibile solo da indirizzi Sapienza; pertanto, o siete in uno qualsiasi dei laboratori Sapienza, oppure potete settare una VPN come descritto all'URL `https://web.uniroma1.it/sbs/sites/default/files/configurareProxy_SBS.pdf`.

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 `http://151.100.17.205/download_from_here/grader.2.tgz` (occorre nuovamente settare la VPN come descritto sopra), 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 `so2.2016.2017.2.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 `so2.2016.2017.2.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 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 `d` o `f`;
- `-gtsize` N, che è vero se il file ha dimensione maggiore o uguale di N bytes (con $N < 2^{32}$);
- `-attrib` i, che è vero se il file ha il setuid bit ($i = 0$), il setgid bit ($i = 1$), lo sticky bit ($i = 2$), o tutti e 3 ($i = 3$).

Quando più test sono presenti, assumere di dover effettuare l'AND degli stessi. Se sono presenti più istanze dello stesso test, considerare solo l'ultima istanza. Il test `-gtsize` N, se applicato ad una directory, risulta sempre vero per qualsiasi valore di N. Non è possibile effettuare altre combinazioni logiche dei test.

Delle azioni, vanno implementate solo le seguenti:

- `-delete`, con lo stesso significato che ha nel `find` di sistema;
- `-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. Se non c'è nessun test, allora tutti i file passano la fase di test.

Si possono manifestare solamente i seguenti errori:

- Non è possibile leggere il contenuto di una delle directory D date nella lista iniziale dei file. Il programma dovrà allora terminare con exit status 10 (senza eseguire alcuna azione), e scrivendo su standard error `Cannot open directory` D `because of` e, dove e è la stringa di sistema che spiega l'errore. Attenzione: se la directory non esiste perché è stata cancellata come effetto di un'azione `delete` ancora in corso, allora occorre limitarsi a procedere con l'esecuzione, senza dare alcun messaggio d'errore.

- 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.

- L'azione è `-delete`, ma non è possibile cancellare un file. Il programma dovrà allora scrivere su standard error `Cannot delete` f `because of` e, dove e è la stringa di sistema che spiega l'errore e f è il nome del file da cancellare. L'esecuzione dovrà poi procedere con i restanti file.

- 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 la system call `system`. 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. C'è una directory `inp.`i anche dentro `check/out.`i: è utile nel caso in cui l'azione sia `-delete` per vedere quale deve essere il risultato (se l'azione è `-print`, sarà semplicemente una copia della directory di input). La directory `check/out_tmp.`i contiene dei log di esempio di `valgrind`.

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 ed un server, che comunichino tramite socket. Più in dettaglio, il client dovrà avere i seguenti argomenti (nell'ordine dato):

- il numero di una porta;
- opzionalmente, un file di testo.

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 accept connections" sullo standard output (tale scrittura dovrà essere seguita da un *flush* dello standard output stesso). Quando arriva un nuova richiesta, la deve servire nel seguente modo: l'input letto dal socket va passato allo standard input del programma `/usr/bin/bc` con opzione `-l`. 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 socket, 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 sul socket 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 tutto quanto letto o dal file dato come argomento, o dallo standard input se il client è stato invocato con un argomento solo. Ogni riga ricevuta dal server va scritta sullo standard output del client stesso.

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 l'argomento richiesto. Il programma dovrà allora terminare con exit status 20 (senza eseguire alcuna azione), e scrivendo su standard error `Usage:` p `port_number [file]`, dove p è il nome del programma stesso.

- Il client viene avviato con 2 argomenti, ma il file f dato come secondo argomento 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 read from file` f `because of` e, dove e è la stringa di sistema che spiega l'errore.

- 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 la system call `system`. 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.

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. La directory `check/out_tmp.`i contiene dei log di esempio di `valgrind`.

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 2 programmi C, `3.read.c` e `3.write.c`, il primo in grado di scrivere ed il secondo in grado di interpretare file binari con un certo formato. Più in dettaglio, entrambi i programmi dovranno prendere come argomento un nome di un file f. In più, `3.read.c` deve anche prendere un valore intero i tra 1 e 9.

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

```
selettore
struttura
```

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

1. un float, un double, e 6 char, il tutto ripetuto 2 volte (senza padding);
2. 2 interi con segno, 2 interi senza segno ed un double, il tutto ripetuto 3 volte;
3. 50 interi senza segno.

Il programma `3.read.c` deve leggere il file f e scriverlo come testo su standard output. Il programma `3.write.c` deve leggere i blocchi come testo da standard input e scriverli in binario sul file f. Per la lettura/scrittura di selettore e struttura *come testo*, valgono le seguenti regole:

1. il selettore è separato 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 float sono in virgola fissa, i double in virgola mobile;
4. sia i float che i double hanno i cifre dopo la virgola;
5. al termine di ogni struttura c'è un a capo.

Si possono manifestare solamente i seguenti errori:

- Il programma `3.read` 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 digits`, dove p è il nome del programma stesso.
- Il programma `3.write` 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 `file`, 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 10 (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` per `3.read` e `write to` per `3.write`.
- Il file f letto da `3.read.c` 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 20.
- 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 la system call `system`. 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.

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 i programmi `./3/3.read` e `./3/3.write` possano essere lanciati, 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`.

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.