Laboratorio di Sistemi Operativi
9 luglio 2014
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (2 punti) Si scriva un comando/pipeline per visualizzare tutte le directory contenute *ricorsivamente* in `/home` *scrivibili* dagli utenti appartenenti al *gruppo* delle directory in questione (*suggerimento*: l'opzione `-R` del comando `ls` consente di eseguire un listing ricorsivo).

   Soluzione:

   ```
   ls -lR /home | grep '^d....w....'
   ```

   oppure

   ```
   ls -lR /home | grep '^d....w'
   ```

2. (2 punti) Cos'è un *here document*? Si faccia un esempio di utilizzo di questo concetto nella shell `bash`.

   Soluzione:

   Un *here document* è un modo per redirigere dell'input multilinea dalla console ad un programma. Sostanzialmente è come se l'input venisse fornito al programma sotto forma di contenuto di un file. Tuttavia il file non esiste fisicamente nel filesystem, ma viene creato "al volo" tramite redirezione dello standard input. Un esempio classico è il seguente:

   ```
   > wc <<delim      # here document
   ?  queste linee formano il contenuto
   ?  del testo
   ?  delim
      2    7    44
   ```

   dove il testo fornito in input al comando `wc` viene digitato alla console dall'utente fintanto che non si incontra una linea contenente il delimitatore `delim` che indica la fine dell'input, ovvero, la fine del documento temporaneo.

3. (3 punti) Sapendo che il comando `date`, quando invocato sulla linea di comando, produce in output una stringa come la seguente:

   ```
   Mon Jul  7 16:46:11 CEST 2014
   ```

   si spieghi qual è l'effetto dei seguenti comandi:

   ```
   val=‘date | cut -d':' -f2‘                    # gli apici esterni sono dei backtick
   echo $val
   ```

   In particolare, se l'output di date è proprio `Mon Jul  7 16:46:11 CEST 2014`, qual è il valore stampato dall'ultimo comando?

   Soluzione:

   I backtick permettono di realizzare la cosiddetta *command substitution*, ovvero, di "catturare" l'output del comando `date | cut -d':' -f2` (che normalmente verrebbe visualizzato sullo schermo del terminale) e di assegnarlo alla variabile `val`. In particolare, nel caso menzionato, l'output del comando saranno i minuti della stringa che rappresenta la data e ora correnti. Infatti il comando `cut` riceve dalla pipeline la data ed ora correnti e la suddivide in campi usando il carattere *due punti* (opzione `-d':'`) come separatore, estraendo il secondo campo (opzione `-f2`), ovvero, i minuti. L'output finale sarà quindi il valore assegnato a `val` in tal modo, ovvero, `46`.

4. (4 punti) Si progetti uno script della shell `triangle.sh` che prenda in input sulla linea di comando un intero positivo e stampi un triangolo di asterischi secondo lo schema dell'esempio seguente:

```
$ ./triangle.sh 5
*
**
***
****
*****
```

Soluzione:

```
i=0

while test $i -lt $1
do
  j=0
  while test $j -le $i
  do
    echo -n '*' # stampo gli asterischi di una riga senza andare a capo
    j=$[$j+1]
  done
  echo # vado a capo alla fine della riga di asterischi
  i=$[$i+1]
done
```

5. Classificare come vere o false le seguenti affermazioni (per quelle false giustificare le risposte):
   (a) (1 punto) La system call `fork()` restituisce al processo figlio il proprio PID, mentre al padre restituisce `NULL`.
   (b) (1 punto) Le system call della famiglia `exec` consentono di sovrascrivere la memoria di un processo, iniziando l'esecuzione di un altro programma.
   (c) (1 punto) Le pipe sono un meccanismo di comunicazione tra processi.
   (d) (1 punto) Le pipe possono essere gestite con diverse politiche (ad esempio, FIFO, LIFO ecc.) e sono bidirezionali (ovvero, con un singolo descrittore di file è possibile leggere e scrivere contemporaneamente nel canale).
   (e) (1 punto) Tramite l'apposita system call è possibile creare insiemi di semafori con un'unica chiamata.
   (f) (1 punto) Le socket possono essere utilizzate soltanto con il dominio Internet, ovvero, fra processi residenti su macchine distinte collegate in rete. Non è possibile utilizzarle per far comunicare processi in esecuzione sulla stessa macchina.

Soluzione:

(a) Falso: la chiamata `fork()` restituisce il PID del figlio al padre e `0` al figlio.

(b) Vero.

(c) Vero.

(d) Falso: le pipe possono essere gestite soltanto con politica FIFO e sono unidirezionali (infatti servono due descrittori di file per leggere e scrivere in una pipe).

(e) Vero.

(f) Falso: usando il dominio UNIX, ad esempio, si possono utilizzare le socket anche fra i processi di una singola macchina, senza necessitare di connessioni di rete.

6. Completare i seguenti frammenti di codice (i `...` indicano le parti mancanti):

   (a) (3 punti)
   ```
   int n;
   printf("Inserisci un numero decimale intero: ");
   scanf("...",...);
   printf("Il numero inserito e': %...\n",...);
   ```

   (b) (3 punti)
   ```
   char *s1="Ciao, mondo!";
   char *s2=...;
   strcpy(...,s1);                                    // copia s1 in s2
   printf("s2: %s\n",...);
   ```

Soluzione:

(a)
```
int n;
printf("Inserisci un numero decimale intero: ");
scanf("%d",&n);
printf("Il numero inserito e': %d\n",n);
```

(b)
```
char *s1="Ciao, mondo!";
char *s2=(char *)malloc(strlen(s1)+1);   // +1 per il terminatore di stringa
strcpy(s2,s1);                           // copia s1 in s2
printf("s2: %s\n",s2);
```

7. Individuare l'errore nei seguenti frammenti di codice:

   (a) (2 punti)
   ```
   filedes = open("nomefile", O_RDONLY);
   write(filedes, "prova", strlen("prova"));
   ```

   (b) (2 punti)
   ```
   filedes=open("prova.txt",O_RDONLY);
   lseek(filedes, (off_t)-40, SEEK_SET);
   ```

Soluzione:

(a) Avendo aperto il file `nomefile` in sola lettura (`O_RDONLY`), la successiva `write` fallirà.

(b) La chiamata a `lseek` cerca di applicare un offset negativo (`-40`) al puntatore alla posizione corrente del file aperto. Ciò non è permesso in quanto equivarrebbe a spostarsi a prima dell'inizio del file stesso (`SEEK_SET`).

8. Si scriva un programma C che prenda in input (come argomento sulla linea di comando) il percorso di un file e stampi a video le seguenti informazioni:

   (a) (1 punto) la dimensione (logica) in byte del file;

   (b) (1 punto) la dimensione del blocco logico del filesystem;

   (c) (3 punti) il numero di byte "sprecati" a causa della frammentazione interna.

   Si ignori la gestione degli eventuali errori. *Suggerimento*: si utilizzi la struttura `struct stat` e le relative system call (i membri della struttura da utilizzare sono: `st_size` per la dimensione logica in byte e `st_blksize` per la dimensione del blocco logico; se si preferisce utilizzare il membro `st_blocks`, si faccia attenzione al fatto che esso rappresenta il numero di blocchi *fisici*, non logici, da 512 byte allocati).

Soluzione:

```
#include <stdio.h>
#include <sys/stat.h>
```

```
#include <sys/types.h>

int main(int argc, char *argv[]) {
  struct stat buf;
  long int dim, blk;

  stat(argv[1], &buf);
  dim=buf.st_size;
  blk=buf.st_blksize;
  printf("Dimensione del file %s: %ld byte\n",argv[1],dim);
  printf("Dimensione del blocco: %ld byte\n",blk);
  printf("Frammentazione interna: ");
  /* Se la dimensione del file (dim)
   * e' un multiplo del blocco (dim%blk==0),
   * allora la frammentazione interna e' 0.
   * Altrimenti e' pari a blk-(dim%blk), ovvero,
   * il residuo di byte non utilizzati (sprecati)
   * nell'ultimo blocco.
   */
  printf("%ld byte\n",((dim%blk==0) ? 0 : blk-(dim%blk)));
  return 0;
}
```