Prova scritta di Sistemi di Elaborazione
Ingegneria delle Telecomunicazioni

Ing. G. Lettieri, Ing. A. Vecchio

17 settembre 2010

1. Supponiamo di avere il seguente programma scritto in parte in Assembler e in parte in C++:

```
.text
.global f1
f1:
  pushl %ebp
  movl  %esp, %ebp
  pushl %esi
  pushl %ebx
  movl  $0, %esi
  movl  8(%ebp), %ebx
l1:
  cmpb  $0, (%ebx)
  je    l2
  incl  %esi
  incl  %ebx
  jmp   l1
l2:
  movl  %esi, %eax
  popl  %ebx
  popl  %esi
  leave
  ret
```

```
#include <stdio.h>
#include <string.h>

const int MAXL = 200;

int f1(const char* s);

void f2(int n, FILE* ff) {
  char ll[MAXL];
  if (ff == 0)
    return;
  while (fgets(ll, MAXL, ff) != 0)
    if(f1(ll) == n)
      fputs(ll, stdout);
}

int main(int argc, char* argv[]){
  FILE *ff;
  int n;
  sscanf(argv[1], "%d", &n);
  ff = fopen(argv[2], "r");
  f2(n, ff);
  return 0;
}
```

(a) Dire cosa viene calcolato dal programma complessivo.

(b) Tradurre la funzione `f2` in Assembler.

2. Scrivere i seguenti programmi in C++, utilizzando le primitive di Unix e la libreria standard del C.

   (a) Un programma `reverse` con zero o più nomi di file come argomenti da riga di comando. I file devono contenere testo, con linee terminate da *a capo* (caratter "`\n`"). Di ogni file il programma mostra sull'uscita standard il contenuto, ma invertendo l'ordine dei caratteri di ciascuna riga (dall'ultimo al primo). La lunghezza massima delle righe non è nota *a priori*. I file che non possono essere aperti in lettura devono essere saltati. Se non viene passato al programma nessun nome di file, il programma legge dal suo ingresso standard.

   (b) Un programma `test-reverse` con due nomi di file da riga di comando. Il programma applica `reverse` due volte al primo file, scrivendo il risultato nel secondo file (che se non esiste viene creato), quindi controlla che i due file siano identici. Per svolgere il suo compito, il programma deve prima creare due processi collegati tramite una pipe, che eseguono il programma `reverse` in modo opportuno. Al termine dei due processi, deve creare un terzo processo che esegue il comando di sistema `cmp`, con argomenti opportuni, e deve stampare "`OK`" o "`ERROR`" sull'uscita standard in base al valore restituito da `cmp`.

   Nota: il comando `cmp` richiede due nomi di file come argomenti da riga di comando e restituisce 0 se il contenuto dei due file è identico, 1 altrimenti.

   Suggerimento: il valore restituito da un processo può essere ottenuto passando l'indirizzo di una variable intera alla funzione `wait`, quindi applicando la funzione `WEXITSTATUS` a tale variabile.