Prova scritta di Sistemi di Elaborazione
Ingegneria delle Telecomunicazioni

Ing. G. Lettieri, Ing. A. Vecchio

18 luglio 2008

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

```
.text
.global f2
f2:     pushl %ebp
        movl  %esp, %ebp
        pushl %edx
        movl  8(%ebp), %eax
        cmpb  $'s, 16(%ebp)
        jne   e1
        addl  12(%ebp), %eax
        jmp   ff
e1:     cmpb  $'d, 16(%ebp)
        jne   e2
        subl  12(%ebp), %eax
        jmp   ff
e2:     cmpb  $'x, 16(%ebp)
        jne   e3
        mull  12(%ebp)
        jmp   ff
e3:     cmpb  $'v, 16(%ebp)
        jne   ff
        movl  $0, %edx
        divl  12(%ebp)
        jmp ff
ff:     popl  %edx
        leave
        ret
```

```
#include <stdio.h>

const int MAXN = 100;

int f2(int x, int y, char c);

void f1(int a[], int m, int b[],
    int n, char c)
{
    int i;
    int k;
    if (n != m)
        return;
    for (i = 0; i < n; i++) {
        k = f2(a[i], b[i], c);
        printf("%d\n", k);
    }
}
```

```
int main(int argc, char *argv[])
{
    int v1[MAXN];
    int v2[MAXN];
    int n = 0;
    int m = 0;
    char c = argv[1][0];
    FILE *l1 = fopen(argv[2], "r");
    FILE *l2 = fopen(argv[3], "r");
    while (fscanf(l1, "%d", &v1[n]) != EOF)
        n++;
    while (fscanf(l2, "%d", &v2[m]) != EOF)
        m++;
    f1(v1, n, v2, m, c);
    return 0;
}
```

(a) Dire cosa viene calcolato dal programma complessivo.

(b) Tradurre la funzione `f1` in Assembler.

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

   (a) Un programma `classifica` con argomenti $intervallo_1$, $intervallo_2$, ... da riga di comando. Ogni $intervallo_i$ deve essere una stringa di tre caratteri della forma "c_1-c_2", dove sia c_1 che c_2 sono singoli caratteri e rappresentano l'intervallo di tutti i caratteri compresi (nell'ordinamento ASCII), tra c_1 e c_2 (inclusi). Almeno un intervallo deve essere fornito.

   Il programma deve leggere l'ingresso standard e produrre in uscita una riga per ogni intervallo. Ogni riga deve avere la forma
   $intervallo_i$`:` *quanti*
   Dove *quanti* è il numero di caratteri letti che cadono nell'intervallo i-esimo. Notare che uno stesso carattere può cadere in più di un intervallo.

   Per esempio, supponiamo di invocare il nostro programma nel seguente modo:
   `./classifica a-c b-d l-n`
   e di inserire in ingresso i seguenti caratteri:
   `abddmn`
   Il programma deve allora produrre la seguente uscita:

   ```
   a-c: 2
   b-d: 3
   l-n: 2
   ```

   (b) Un programma `parclass` con argomenti *num*, *intervallo*, *filein* e *fileout*. Il programma deve creare *num* processi figli, ciascuno dei quali esegue `classifica` sullo stesso *intervallo*. Tutti i processi figli devono leggere dalla stessa pipe e scrivere (in modalità APPEND) su *fileout*. Il processo padre deve leggere il file *filein* e inviare tutti i caratteri letti sulla pipe da cui leggono tutti i processi figli. Alla fine, il processo padre attende la terminazione dei figli e termina esso stesso. **Nota**: su *fileout* verrà scritta una serie di righe "*intervallo*: *quanti*". Ogni riga è prodotta da uno dei figli e *quanti* è il numero di caratteri che cadevano in *intervallo* tra quelli visti da quel figlio.