[2011-12] Pipe

Si deve realizzare una semplice calcolatrice in grado di sommare una lista di numeri, secondo le seguenti specifiche:

1. La calcolatrice è in un ciclo infinito: continua a prendere input e dare output finché non viene interrotta
2. prende l'input da una pipe 'calcPipeIn' e manda i risultati su una seconda pipe 'calcPipeOut'. Le pipe vengono create dalla calcolatrice stessa quando viene eseguita
3. le espressioni sono semplici somme, ad esempio 10 + 15 + 280. Vengono inviate su calcPipeIn come sequenze di char terminate da #. Ad esempio 10 + 15 + 280 viene inviata come '10+15+280#', ovvero 10 byte **senza il terminatore di stringa e senza spazi**
4. il risultati vengono inviati sempre come sequenze di char separati da '#'. Se, ad esempio, inviamo '1+2#3+4#' su calcPipeIn ci aspetteremo '3#7#' su calcPipeOut

Viene fornito un programma di test che invia NT espressioni e controlla i risultati. Per vedere il formato di invio dei dati create le pipe da terminale e stampate il contenuto della pipe di output. Poi eseguite il test da un altro terminale. Ecco un esempio:

```
$ mkfifo calcPipeIn calcPipeOut
$ cat calcPipeIn
39+23+91+95+60+27+53+56#99+90+85+35#69+21+37+9#89+77+23+14+49+93+16#37#96+21+24+19+13+1
$
```

E questo è il corrispondente ouput del file di test

```
$ ./calc_test
=== INIZIO TEST ===
Espressione 0 composta da 8 numeri: 39 23 91 95 60 27 53 56
  somma: 444
  leggo dalla pipe ... :FAIL! timeout
Espressione 1 composta da 4 numeri: 99 90 85 35
  somma: 309
  leggo dalla pipe ... :FAIL! timeout
Espressione 2 composta da 4 numeri: 69 21 37 9
  somma: 136
  leggo dalla pipe ... :FAIL! timeout
Espressione 3 composta da 7 numeri: 89 77 23 14 49 93 16
  somma: 361
  leggo dalla pipe ... :FAIL! timeout
Espressione 4 composta da 1 numeri: 37
  somma: 37
  leggo dalla pipe ... :FAIL! timeout
Espressione 5 composta da 8 numeri: 96 21 24 19 13 19 80 40
  somma: 312
  leggo dalla pipe ... :FAIL! timeout
=== TEST FALLITO ===
```

Il sorgente del programma di test è il seguente

```
1  #include <stdlib.h>
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <fcntl.h>
5  #include <stdbool.h>
6  #include <stdio.h>
7  #include <string.h>
8  #include <stdarg.h>
```

```
 9  #include <errno.h>
10
11  #define PIPE_IN "calcPipeIn"
12  #define PIPE_OUT "calcPipeOut"
13  #define MAXEXP 100
14  #define NT 6
15  #define DEBUG 1
16
17  die(char * s) {
18      perror(s);
19      exit(1);
20  }
21
22  // output abilitato se DEBUG == 1
23  void d_print(char *s, ...) {
24      va_list ap;
25
26      if (DEBUG) {
27          va_start(ap, s);
28          vprintf(s,ap);
29          va_end(ap);
30      }
31  }
32
33  main() {
34      int pin,pout,nums,n,i,j,k,r,fail=false,rread;
35      char sn[MAXEXP],ris[MAXEXP];
36
37      // apre le pipe
38      if( ( pin=open(PIPE_IN,O_RDWR) ) < 0 ||   ( pout=open(PI
39          die("Errore apertura pipe");
40
41      srand(time(NULL)); // inizializza il generatore random
42
43      d_print("=== INIZIO TEST ===\n");
44
45      // genera NT espressioni le invia e controlla il risult
46      for(i=0;i<NT;i++) {
47          r=0; // azzera la somma (per il test)
48
49          nums = (int) ((float)rand() / RAND_MAX * 9) + 1; //
50
51          d_print("Espressione %i composta da %i numeri: ",i,
52
53          // genera i nums numeri e li invia come stringhe se
54          // l'ultimo numero e' terminato con #
55          for(j=0;j<nums;j++) {
56              // sceglie un numero a caso
57              n = (int) ((float)rand() / RAND_MAX * 100);
58              d_print("%i ",n);
59              r += n; // aggiorna la somma (per il test succe
60              sprintf(sn,"%i",n); // crea la stringa
61              write(pin,sn,strlen(sn)); // la invia sulla pip
```

```
62              if (j==nums-1) // e' l'ultimo numero?
63                  write(pin,"#",1); // termina l'espressione
64              else
65                  write(pin,"+",1); // aggiunge il simbolo +
66          }
67          d_print("\n  somma: %i",r);
68
69          // controlliamo il risultato dalla pipe
70          d_print("\n  leggo dalla pipe ... :"); fflush(stdou
71          k=0;
72
73          // leggo un risultato. La lettura e' non bloccante
74          while(k<MAXEXP-1 && (rread=read(pout,&ris[k],1))>0
75
76          // se per caso non ha letto nulla ritenta dopo un s
77          if (rread < 0 && errno ==EAGAIN) {
78              sleep(1);
79              while(k<MAXEXP-1 && (rread=read(pout,&ris[k],1)
80          }
81
82
83          if (rread < 0 && errno ==EAGAIN) {
84              // anche al secondo tentativo non ho letto null
85              d_print("FAIL! timeout\n");
86              fail = true;
87          } else if (atoi(ris) == r)
88              // ho letto la somma corretta
89              d_print("OK\n");
90          else {
91              // ho letto un valore errato, stampo la stringa
92              ris[k+1] ='\0';
93              d_print("FAIL! Ho letto %s\n",ris);
94              fail = true;
95          }
96      }
97
98      if (fail) {
99          d_print("=== TEST FALLITO ===\n");
100         exit(1);
101     }else{
102         d_print("=== TEST SUPERATO ===\n");
103         exit(0);
104     }
105 }
```

Comments: 1

Leave a reply »

Leonid

December 4th, 2013 at 23:25

Un anno dopo l'ultimo commento scrivo pure io la mia soluzione, io l'ho fatta un po diversamente da ciò visto nelle altre soluzioni, ho provato anche con il test proposto ed ho ottenuto questo risultato:
[pre]
=== INIZIO TEST ===
Espressione 0 composta da 8 numeri: 62 92 60 41 44 11 35 4
somma: 349
leggo dalla pipe … :OK
Espressione 1 composta da 2 numeri: 26 88
somma: 114
leggo dalla pipe … :OK
Espressione 2 composta da 2 numeri: 46 83
somma: 129
leggo dalla pipe … :OK
Espressione 3 composta da 3 numeri: 52 41 44
somma: 137
leggo dalla pipe … :OK
Espressione 4 composta da 6 numeri: 30 18 55 72 40 6
somma: 221
leggo dalla pipe … :OK
Espressione 5 composta da 3 numeri: 84 68 4
somma: 156
leggo dalla pipe … :OK
=== TEST SUPERATO ===
[/pre]

questo è il codice:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  //mkfifo, open
6  #include <sys/types.h>
7  #include <sys/stat.h>
8
9  //open
10 #include <fcntl.h>
11
12 //read, write
13 #include <unistd.h>
14
15 //signal
16 #include <signal.h>
17
18 #define INPIPE  "calcPipeIn"
19 #define OUTPIPE "calcPipeOut"
20
21 void safeClose(){
```

```
22      unlink(INPIPE);
23      unlink(OUTPIPE);
24      exit(1);
25  }
26
27  int main(){
28      //mi proteggo, cancello le pipe in caso di rimozione for
29      signal(SIGINT, safeClose);
30
31      //creazione delle pipe
32      mkfifo(INPIPE, 0666);
33      mkfifo(OUTPIPE, 0666);
34
35      int pd0, pd1; //descritori
36
37      //cerco di aprire le pipe
38      if(((pd0 = open(INPIPE, O_RDONLY)) < 0) || ((pd1 = open
39          perror("Errore durante apertura pipe");
40          exit(1);
41      }
42
43      char messaggio[100];
44      int number;
45      char op;
46
47      int i = 0;
48
49      while(read(pd0, &messaggio[i], 1)){
50          if((i>=100) || (messaggio[i++] == '#')){
51              i = 0;
52              //Qui ho la stringa da fare il parsing
53              do{
54                  sscanf(messaggio, "%d%c%s", &number, &op, me
55                  i += number;
56              }while(op != '#');
57
58              sprintf(messaggio, "%d#", i);
59
60              write(pd1, messaggio, strlen(messaggio));
61              i = 0;
62          }
63      }
64
65      unlink(INPIPE);
66      unlink(OUTPIPE);
67      return 0;
68  }
```

spero che sia sulla strada giusta..

Leonid

Leave a Reply

Name *

Mail * (will not be published)

Website

Comment

Submit Comment