[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 calcPipeOutmkfifo
$ 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+19+80+40#
$
```

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

```
#include
#include &lt;sys/types.h&gt;
#include &lt;sys/stat.h&gt;
#include
#include
#include
#include
#include
#include

#define PIPE_IN "calcPipeIn"
#define PIPE_OUT "calcPipeOut"
#define MAXEXP 100
#define NT 6
#define DEBUG 1

die(char * s) {
    perror(s);
    exit(1);
}

// output abilitato se DEBUG == 1
void d_print(char *s, ...) {
    va_list ap;

    if (DEBUG) {
        va_start(ap, s);
        vprintf(s,ap);
        va_end(ap);
    }
}

main() {
    int pin,pout,nums,n,i,j,k,r,fail=false,rread;
    char sn[MAXEXP],ris[MAXEXP];

    // apre le pipe
    if( ( pin=open(PIPE_IN,O_RDWR) ) &lt; 0 ||  ( pout=open(PIPE_OUT,O_RDWR | O_NONBLOCK) ) &lt; 0 )
        die("Errore apertura pipe");

    srand(time(NULL)); // inizializza il generatore random

    d_print("=== INIZIO TEST ===\n");

    // genera NT espressioni le invia e controlla il risultato ricevuto
    for(i=0;i0 &amp;&amp; ris[k] != '#') k++;

        // se per caso non ha letto nulla ritenta dopo un secondo
        if (rread &lt; 0 &amp;&amp; errno ==EAGAIN) {
            sleep(1);
            while(k0 &amp;&amp; ris[k] != '#') k++;
        }

        if (rread &lt; 0 &amp;&amp; errno ==EAGAIN) {
            // anche al secondo tentativo non ho letto nulla
            d_print("FAIL! timeout\n");
            fail = true;
        } else if (atoi(ris) == r)
            // ho letto la somma corretta
            d_print("OK\n");
        else {
            // ho letto un valore errato, stampo la stringa letta dalla pipe
            ris[k+1] ='\0';
            d_print("FAIL! Ho letto %s\n",ris);
            fail = true;
        }
    }

    if (fail)
        d_print("=== TEST FALLITO ===\n");
    else
        d_print("=== TEST SUPERATO ===\n");
}
```