LabSO2022

ESERCIZI SVOLTI/PROPOSTI - II

Nota: è possibile siano presenti dei refusi, per cui valgono le informazioni fornite a lezione, mentre in alcuni casi sono presenti delle semplificazioni a puro titolo di esempio.

Simulazione piping

Realizzare un tool che esegua due comandi (con possibili argomenti) “in piping” avendo un primo parametro “token” da usare come separatore, ad esempio:
./piping pipe ls -alh /tmp pipe wc -l
equivale a:
ls -alh /tmp | wc -l

SOLUZIONE:

```
/*
    // piping.c
    // gcc piping.c -o piping

    ./piping <next> <cmd1> <args_cmd1...> <next> <cmd2> <args_cmd2...>

*/

#define DEBUG 0

#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define READ 0
#define WRITE 1
#define MAXITM 32
#define MAXLEN 32

int main (int argc, char *argv[]) {
    int fd[2], f, a=1, i1=0, i2=0;     // pipe, fork, argv counter, cmd1 and cm2 counter
    char next[MAXLEN];                 // to store "next" token
    char *cmd1[MAXLEN], *cmd2[MAXLEN]; // to store commands with arguments
    if (argc<5) { fprintf(stderr, "?Syntax error, missing arrguments\n"); exit(3); };
    strcpy(next, argv[a++]); // store "next" token
    // first command:
    cmd1[i1++]=argv[a++];
    while (a<argc && strcmp(next, argv[a])!=0) {
        cmd1[i1++]=argv[a++];
    };
    cmd1[i1++] = NULL;
    if (a++==argc) { fprintf(stderr, "?Syntax error, missing pipe token\n"); exit(4); };
    // second command:
    cmd2[i2++]=argv[a++];
```

```
    while (a<argc && strcmp(next, argv[a])!=0) {
        cmd2[i2++]=argv[a++];
    };
    cmd2[i2++] = NULL;
    // dump data:
    if (DEBUG) {
        fprintf(stderr, "a=%d, i1=%d, i2=%d\n", a, i1, i2);
        for (a=0; a<i1; a++) { fprintf(stderr, "cmd1[%d]=%s\n", a, cmd1[a]); };
        for (a=0; a<i2; a++) { fprintf(stderr, "cmd2[%d]=%s\n", a, cmd2[a]); };
    };
    pipe(fd); // Create an unnamed pipe
    f = fork(); if (f<0) { fprintf(stderr, "?Fork error"); exit(2); };
       if (f > 0) { // Parent, writer
              close(fd[READ]); // Close unused end
              dup2(fd[WRITE], 1); // Duplicate used end to stdout
              close(fd[WRITE]); // Close original used end
              execvp(cmd1[0], (char *const *)cmd1); // Execute writer program
              fprintf(stderr,
                  "?Error running command '%s' (#%d %s)\n", cmd1[0], errno, strerror(errno)
              );
       } else { // Child, reader
              close(fd[WRITE]); // Close unused end
              dup2(fd[READ], 0); // Duplicate used end to stdin
              close(fd[READ]); // Close original used end
              execvp(cmd2[0], (char *const *)cmd2); // Execute writer program
              fprintf(stderr,
                  "?Error running command '%s' (#%d %s)\n", cmd2[0], errno, strerror(errno)
              );
       }
}
```

Gestione figli con eventuale passaggio di dati

Realizzare un tool “strings” che accetti come argomento un percorso di un file su disco seguito da un elenco di “n” stringhe generando un figlio - che resta in attesa di un segnale SIGUSR2 - per ciascuna e restando poi esso stesso in ascolto di un segnale SIGUSR1: quando lo riceve deve leggere dal file la prima riga che ci si aspetti contenga un intero e inviare SIGUSR2 al figlio corrispondente, il quale deve stampare a video la stringa corrispondente e terminare.

Esempio:
./strings /tmp/child.txt Prima Seconda
Si generano tre figli (individuabili come 1, 2 e 3). Se si scrive su /tmp/child.txt il valore 3 e poi si invia SIGUSR1 al processo principale questo deve inviare al terzo figlio un SIGUSR2 il quale stampa a video la sua stringa (“Terza con spazio”) e poi termina. Si può rifare lo stesso scrivendo 1 e poi 2 nel file di testo.

I vari casi d’errore devono essere gestiti con informazioni per l’utente a video. Una volta terminati tutti i figli anche il processo principale termina.

VARIANTE: nel file di testo si scrive non solo l’indice ma anche una stringa (ad esempio “3,testo di esempio”) e il processo principale deve inviarla (non deve essere il figlio a leggerla dal file) al figlio corrispondente il quale la stampa insieme alla sua corrispondente come sopra. Nell’esempio di prima il terzo figlio dovrebbe stampare due righe: “Terza con spazio” e poi anche “testo di esempio”

SOLUZIONE:

```
/*

    Gestione figli con eventuale passaggio di dati

    TODO: aggiungere controllo errori nel caso un figlio sia già terminato
    (si può ad esempio verificare se risponde prima di inviare SIGUSR2)

*/

#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/wait.h>
```

```
#define DEBUG 1

#define MAXLEN 32
#define MAXCHL 10
#define READ 0
#define WRITE 1

int  pidmain;
char filename[MAXLEN];
int  pids[MAXCHL], idx;
char string[MAXLEN];
int  pipes[MAXCHL][2]; // extra
int  flag=0;

void handler(int signum) {
    int fd, dat;
    char buf[MAXLEN];
    if (getpid()==pidmain) {
        if (signum == SIGUSR1) {
            fd = open(filename, O_RDONLY);
            if (fd>0) {
                dat = read(fd, buf, MAXLEN-1);
                idx = atoi(buf);
                // idx is child's index, buf the entire text
                write(pipes[idx-1][WRITE], buf, strlen(buf)); // extra
                kill(pids[idx-1], SIGUSR2);
            } else {
                fprintf(stderr, "?File not read.\n");
            }
        };
    } else {
        if (signum == SIGUSR2) {
            printf("Child #%d (%d): '%s'\n", idx, getpid(), string);
            flag=1;
        };
    };
}

int main(int argc, char **argv) {
    char str[MAXLEN]; int s; // extra
    int c, n, f, p, i;
    if (argc<3) { fprintf(stderr, "?Syntax error\n"); exit(2); };
    strcpy(filename, argv[1]); n=argc-2; p=getpid(); pidmain=p;
    printf("filename='%s', children: %d, pid=%d\n", filename, n, p);
    for (c=0; c<n; c++) {
        i=c+1;
        if (pipe(pipes[c])<0) { fprintf(stderr, "?Pipe error\n"); exit(5); };
        f=fork();
        if (f<0) { fprintf(stderr, "?Fork error (%d)\n", i); exit(3); };
        if (f==0) {
            idx=i; p=getpid();
            close(pipes[c][WRITE]);
            strcpy(string, argv[i+1]);
            if (DEBUG) { fprintf(stderr, "Child #%d. pid=%d, string='%s'\n", idx, p, string);
};
            signal(SIGUSR2, handler);
            s=read(pipes[c][READ], str, MAXLEN-1); // extra
```

```
            while (flag==0);
            printf("Child #%d (%d): '%s' (extra)\n", idx, p, str); // extra
            exit(0);
        } else {
            close(pipes[c][READ]);
            pids[c]=f;
        };
    };
    signal(SIGUSR1, handler);
    while (wait(NULL)>0);
    return 0;
}
```