Esercitazione di Venerdì 11 Maggio 2018

1. (pipe-newSenzascrittore1.c) Partendo dall'esercizio pipe-newSenzascrittore.c mostrato a lezione, eliminare le chiusure dei lati delle pipe nel figlio e nel padre (ad esempio commentandole) e quindi verificare che alla morte del figlio (che è lo scrittore) il padre invece che uscire dal ciclo while di lettura dalla pipe (per effetto del valore di ritorno 0) rimane bloccato nella lettura e quindi non torna il prompt dei comandi e quindi bisognerà abortire il comando utilizzando il ^C.

```
soELab@Lica02:~/pipe$ cat pipe-newSenzascrittore1.c
/* FILE: pipe-newSenzascrittore1.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#define MSGSIZE 5

int main (int argc, char **argv)
{       int pid, j, piped[2];                   /* pid per fork, j
per indice, piped per pipe */
        char mess[MSGSIZE];                     /* array usato dal
figlio per inviare stringa al padre */
        char inpbuf [MSGSIZE];                  /* array usato dal
padre per ricevere stringa inviata dal figlio: N.B: si poteva usare
sempre mess, tanto il padre e il figlio agiscono sulla loro copia
delle variabili! */
        int pidFiglio, status, ritorno;          /* per wait padre
*/

if (argc != 2)
{   printf("Numero dei parametri errato\n");
    exit(1);
}
/* si crea una pipe */
if (pipe (piped) < 0 )  { exit (2); }
if ((pid = fork()) < 0)  { exit (3); }
else if (pid == 0)
{
/* figlio */
int fd;
/* COMMENTIAMO LA CHIUSURA close (piped [0]); */  /* il figlio
AVREBBE DOVUTO CHIUDERE il lato di lettura */
if ((fd = open(argv[1], O_RDONLY)) < 0)
{   printf("Errore in apertura file %s\n", argv[1]);
    exit(4);
}

printf("Figlio %d sta per iniziare a scrivere una serie di messaggi,
ognuno di lunghezza %d, sulla pipe dopo averli letti dal file passato
come parametro\n", getpid(), MSGSIZE);
/* IL FIGLIO TERMINA ===> PIPE SENZA SCRITTORE */
exit (0);
}
```

```
/* padre */
/* COMMENTIAMO LA CHIUSURA close (piped [1]); */ /* il padre AVREBBE DOVUTO CHIUDERE il lato di scrittura */
printf("Padre %d sta per iniziare a leggere i messaggi dalla pipe\n", getpid());
j =0; /* il padre inizializza la sua variabile j per verificare quanti messaggi ha mandato il figlio */
while (read ( piped[0], inpbuf, MSGSIZE))
    {
        /* dato che il figlio gli ha inviato delle stringhe, il padre le puo' scrivere direttamente con una printf */
        printf ("%d: %s\n", j, inpbuf);
        j++;
     }
printf("Padre %d letto %d messaggi dalla pipe\n", getpid(), j);
/* padre aspetta il figlio */
pidFiglio = wait(&status);
if (pidFiglio < 0)
{
      printf("Errore wait\n");
      exit(5);
}
if ((status & 0xFF) != 0)
        printf("Figlio con pid %d terminato in modo anomalo\n", pidFiglio);
else
{
       ritorno=(int)((status >> 8) & 0xFF);
       printf("Il figlio con pid=%d ha ritornato %d\n", pidFiglio, ritorno);
}
```

Verifichiamo cosa cambia rispetto a prima:

```
soELab@Lica02:~/pipe$ pipe-newSenzascrittore1 input.txt
Padre 22575 sta per iniziare a leggere i messaggi dalla pipe
Figlio 22576 sta per iniziare a scrivere una serie di messaggi, ognuno di lunghezza 5, sulla pipe dopo averli letti dal file passato come parametro
```

<== NON torna il prompt, dato che la prima read eseguita dal padre risulta bloccante e quindi per far terminare il processo padre dovremo usare CRTL-C (situazione analoga al processo lungpipe1). Questo succede perché i processi (padre e figlio) rimangono entrambi potenziali lettori/scrittori della pipe e quindi quando termina il figlio, per il Sistema Operativo c'è ancora un processo (che è il padre) che potrebbe scrivere, ma che chiaramente non scrive e quindi si blocca sulla read!

Abortiamo il programma e quindi facciamo terminare il processo padre (N.B.: il figlio verrà ereditato da init che provvederà ad eliminarlo dal sistema eliminandone il descrittore di processo, questo lo vedremo più avanti):

```
^C
soELab@Lica02:~/pipe$
```

2. (5Giu15.c) Esame del 5 Giugno 2015 (seconda prova in Itinere, quindi solo parte C): La parte in C accetta un numero variabile di parametri (maggiore o uguale a 2, da controllare) che rappresentano M nomi assoluti di file F1...FM.
   Il processo padre deve generare M processi figli (P0 … PM-1): ogni processo figlio è associato al corrispondente file Fj. Ognuno di tali processi figli deve creare a sua volta un processo nipote (PP0 … PPM-1): ogni processo nipote PPj esegue concorrentemente e deve, usando in modo opportuno il comando tail di UNIX/Linux, leggere l'ultima linea del file associato Fi.
   Ogni processo figlio Pj deve calcolare la lunghezza, in termini di valore intero (lunghezza), della linea scritta (escluso il terminatore di linea) sullo standard output dal comando tail eseguito dal processo nipote PPj; quindi ogni figlio Pj deve comunicare tale lunghezza al padre. Il padre ha il compito di ricevere, rispettando l'ordine inverso dei file, il valore lunghezza inviato da ogni figlio Pj che deve essere riportato sullo standard output insieme all'indice del processo figlio e al nome del file cui tale lunghezza si riferisce.
   Al termine, ogni processo figlio Pj deve ritornare al padre il valore di ritorno del proprio processo nipote PPi e il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato.

La soluzione si trova alla URL
http://didattica.agentgroup.unimore.it/didattica/SOeLab/SoluzioniCompiti/5Giu15/5Giu15.c

3. (8Giu16.c) Esame del 8 Giugno 2016 (consideriamo la sola parte C): La parte in C accetta un numero variabile N+1 di parametri (con N maggiore o uguale a 4) che rappresentano i primi N nomi di file (F0, F1, ... FN-1), mentre l'ultimo rappresenta un numero intero (H) strettamente positivo e minore di 255 (da controllare) che indica la lunghezza in linee dei file: infatti, la lunghezza in linee dei file è la stessa (questo viene garantito dalla parte shell e NON deve essere controllato).
   Il processo padre deve, per prima cosa, inizializzare il seme per la generazione random di numeri (come illustrato nel retro del foglio) e deve creare un file di nome "/tmp/creato" (Fcreato). Il processo padre deve, quindi, generare N processi figli (P0 … PN-1): i processi figli Pi (con i che varia da 0 a N-1) sono associati agli N file Fk (con k= i+1). Ogni processo figlio Pi deve leggere le linee del file associato Fk sempre fino alla fine. I processi figli Pi e il processo padre devono attenersi a questo schema di comunicazione: per ogni linea letta, il figlio Pi deve comunicare al padre la lunghezza della linea corrente compreso il terminatore di linea (come int); il padre usando in modo opportuno la funzione mia_random() (riportata nel retro del foglio) deve individuare in modo random, appunto, quale lunghezza (come int) considerare fra quelle ricevute, rispettando l'ordine dei file, da tutti i figli Pi; individuata questa lunghezza, usando sempre in modo opportuno la funzione mia_random() deve individuare un intero che rappresenterà un indice per la linea della lunghezza considerata; il padre deve comunicare indietro a tutti i figli Pi tale indice: ogni figlio Pi ricevuto l'indice (per ogni linea) deve controllare se è ammissibile per la linea corrente e in tal caso deve scrivere il carattere della linea corrente, corrispondente a tale indice, sul file Fcreato, altrimenti non deve fare nulla e deve passare a leggere la linea successiva.
   Al termine, ogni processo figlio Pi deve ritornare al padre il valore intero corrispondente al numero di caratteri scritti sul file Fcreato (sicuramente minore di 255); il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato.
   Chiamata alla funzione di libreria per inizializzare il seme:

```
#include <time.h>
srand(time(NULL));
```

   Funzione che calcola un numero random compreso fra 0 e n-1:

```
#include <stdlib.h>

int mia_random(int n)
{
int casuale;
casuale = rand() % n;
return casuale;
}
```

La soluzione si trova alla URL
http://didattica.agentgroup.unimore.it/didattica/SOeLab/SoluzioniCompiti/8Giu16/8Giu16.c