SISTEMI OPERATIVI e
LABORATORIO DI SISTEMI OPERATIVI
(A.A. 07-08) – 11 MARZO 2008

Esercizio

Si realizzi un programma **concorrente** per UNIX che deve avere una parte in **Bourne Shell** (già realizzata) e una parte in **C**.

Parte in Linguaggio C

La parte in C accetta un numero variabile **pari** di parametri (2*N) che rappresentano le seguenti informazioni: i primi N parametri rappresentano nomi di file F1...FN, mentre i secondi N parametri rappresentano delle stringhe S1...SN.
Il processo padre deve generare **N processi figli** (**P1** … **PN**): ogni processo figlio è associato al corrispondente file **Fi** e alla corrispondente stringa **Si**. Ognuno di tali processi figli deve creare a sua volta un **processo nipote (PP1 … PPN**): ogni processo nipote **PPi** esegue concorrentemente andando a verificare se nel file associato **Fi** esiste almeno una occorrenza della stringa associata **Si** usando in modo opportuno il comando ***grep*** di UNIX/Linux.
Ogni processo figlio **Pi**, sulla base del valore di ritorno del nipote **PPi** di successo o insuccesso della grep per la stringa **Si** associata, deve comunicare al padre rispettivamente la stringa "TROVATA" o "NON TROVATA".
Per ogni figlio, il padre ha il compito di stampare su standard output, **rispettando l'ordine dei file**, **la stringa ricevuta** riportando sia il nome del file che il valore della stringa.
Al termine, ogni processo figlio **Pi** deve ritornare al padre il valore di ritorno del proprio processo nipote e il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato.

```c
/* Soluzione della Prova d'esame dell'11 Marzo 2008 - Parte C*/
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>

typedef int pipe_t[2];

int main(int argc, char **argv) {

	/* -------- Variabili locali ---------- */
	int pid;	/* process identifier per le fork() */
	int N;		/* numero di file passati sulla riga di comando */
	int rc;   /* return value che il figlio passa al padre alla fine */
	int status;		/* variabile di stato per la wait */
	pipe_t *piped;		/* array dinamico di pipe descriptors */
	int i, j;		/* indici per i cicli */
	char stringa[12];	/* variabile dove il padre mette la sua stringa
				TROVATA/NON TROVATA */
	/* ------------------------------------ */

	/* Controllo sul numero di parametri */
	if( (argc<3) /* Meno di due parametri */ || ((argc-1)%2) /* In numero
	dispari*/)
	{
		printf("Errore nel numero dei parametri: devono essere in
		numero pari!\n");
		exit(-1);
	}

	N = (argc - 1)/2;

	/* Allocazione dell'array di N pipe descriptors*/
	piped = (pipe_t *) malloc (N*sizeof(pipe_t));
	if(piped == NULL)
	{
		printf("Errore nella allocazione della memoria");
		exit(-2);
	}

	/* Creazione delle N pipe padre-figli */
	for(i=0; i<N; i++)
	{
		if(pipe(piped[i]) < 0)
		{
			printf("Errore nella creazione della pipe");
			exit(-3);
		}
	}

	/* Ciclo di generazione dei figli */
	for(i=0; i<N; i++)
	{
		if( (pid = fork()) < 0)
		{
			printf("Errore nella fork.");
```

```
        exit(-4);
}

if(pid == 0) /* codice del figlio */
{
        /* Chiusura dei lati non utilizzati delle pipe */
        for(j=0; j<N; j++)
        {
                close(piped[j][0]);
                if(j!=i)
                        close(piped[j][1]);
        }

        /* Creazione del processo nipote */
        if( (pid = fork()) < 0)
        {
                printf("Errore nella fork.");
                exit(-4);
        }

        if(pid == 0) /* codice del nipote */
        {
           /* Chiusura dell'unico lato di pipe rimasto aperto */
                close(piped[i][1]);

                printf("Figlio %d cerca nel file %s la stringa
                %s\n", getpid(), argv[i+1], argv[i+N+1]);

                /* Ridirezione dello standard input su /dev/null
                (per evitare messaggi di grep a video)*/
                close(1);
                open("/dev/null", O_WRONLY);

                /* Il nipote diventa il comando grep */

                execl("/bin/grep", "grep", argv[i+N+1],
                argv[i+1], (char *)0);

                /* attenzione all'ordine dei parametri nella
                esecuzione della grep: prima stringa e poi file e
                quindi terminatore della lista. Il file si trova
                usando l'indice i incrementato di 1 (cioe' per il
                primo processo i=0 il file e' argv[1]), mentre la
                stringa la si trova usando l'indice i incrementato
                di 1 e di N (cioe' per il primo processo i=0 la
                stringa file e' argv[1+N])  */

                /* Non si dovrebbe mai tornare qui!!*/
                exit(1);
                /* torno un valore diverso da zero per indicare
                insuccesso*/
        }

        /* codice del figlio */

        /* Aspetto la fine del nipote/grep */
        wait(&status);
```

```
                        /* recupero il suo codice di ritorno */
                        rc = (status >> 8) & 0xFF;

                        if(rc == 0) /* stringa trovata */
                                strcpy(stringa, "TROVATA");
                        else
                                strcpy(stringa, "NON TROVATA");

                        /* comunico la stringa al padre. La lunghezza va
                        aumentata del terminatore, cosi non devo terminare la
                        stringa letta nel padre. */
                        write(piped[i][1], stringa, strlen(stringa) + 1);

                        exit(rc); /* fine codice del figlio */
                }
        }

        /* Codice del padre */
        for(i=0; i<N; i++)
                close(piped[i][1]);

        /* Il padre recupera le informazioni dai figli in ordine di indice */
        for(i=0; i<N; i++)
        {
                /* il padre recupera al massimo 12 caratteri dalla pipe che
                comprendono sempre la stringa inviata dal figlio e il
                terminatore */
                read(piped[i][0], stringa, 12);
                printf("Il figlio di indice %d ha analizzato il file %s,
                cercando la stringa %s ed ha ritornato il seguente
                messaggio:%s\n", i, argv[i+1], argv[i+N+1], stringa);
        }

        /* Il padre aspetta i figli in ordine di indice */
        for(i=0; i<N; i++)
        {
                pid = wait(&status);
                printf("Il figlio con pid=%d ha ritornato %d\n", pid,
                (int)((status >> 8) & 0xFF));
        }
}
```