SISTEMI OPERATIVI e
LABORATORIO DI SISTEMI OPERATIVI
(A.A. 05-06) – 21 Luglio 2006

Esercizio

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

La parte in Shell deve prevedere due parametri: il primo deve essere il **nome assoluto di un direttorio** che identifica una gerarchia (**G**) all'interno del file system, mentre il secondo parametro deve essere considerato un numero **K** strettamente positivo. Il programma deve cercare nella gerarchia **G** specificata (compresa la radice) tutti i direttori che contengono almeno **un** file la cui lunghezza in linee sia esattamente uguale a **K**, se K è un numero pari, o esattamente uguale a **K+1**, se K è un numero dispari. Si riporti il nome assoluto di tali direttori sullo standard output. In ogni direttorio trovato, si deve invocare la parte in C, passando come parametri i nomi dei file trovati (**F0... FM-1**/**FN-1**) che soddisfano la condizione precedente.

La parte in C accetta un numero variabile di parametri che rappresentano nomi di file **F0...FM-1**/**FN-1**: la lunghezza in linee di ognuno dei file è un numero pari (questo viene garantito dalla parte shell e NON deve essere controllato). Il processo padre deve generare **N/M processi figli** (**P1** … **PN/M**): ogni processo figlio è associato ad uno dei file **Fi/j**. Ognuno di tali processi figli deve creare un **processo nipote** e quindi deve creare un file il cui nome (**FCreatoi/j**) risulti dalla concatenazione del nome del file associato (**Fi/j**) con la stringa "pari/dispari". La **coppia** figlio e nipote esegue concorrentemente leggendo, rispettivamente, il figlio/nipote le linee pari/dispari del file associato **Fi/j** e il nipote/figlio le linee dispari/pari del file associato **Fi/j** (si ricordi che ogni file **Fi/j** ha una lunghezza pari di linee!). Il processo nipote, dopo la lettura di ogni linea dispari/pari, spedisce al processo figlio la lunghezza di tale linea Ld/p; il processo figlio, dopo la lettura di ogni linea pari/dispari, calcola la lunghezza della propria linea Lp/d e riceve la lunghezza Ld/p dal nipote: il processo figlio scrive la linea pari/dispari corrente sul file **FCreatoi/j se e solo se** Lp/d risulta minore strettamente/maggiore strettamente/uguale/diversa/minore/maggiore di Ld/p. Ogni processo figlio deve ritornare al padre il numero di linee pari/dispari scritte/non scritte sul file **FCreatoi/j**.
Il padre, dopo che i figli sono terminati, deve stampare su standard output i PID di ogni figlio con il corrispondente valore ritornato.

```
#!/bin/sh
# Soluzione dell'esame del 21 Luglio 2006
# file di controllo parametri
# SINTASSI: $0 dirass K
#

# controllo numero di parametri
if test $# -lt 2
then
       echo "Devi specificare almeno due parametri: dirass K "
       exit 1
fi


#controllo direttorio assoluto
case $1 in
/*)    if test ! -d $1 -o ! -x $1
       then
              echo "Non direttorio o non accessibile"
              exit 2
       fi;;

*)     echo "Nome non assoluto di direttorio"
       exit 2;;
esac


#controllo secondo parametro K numero strettamente positivo
expr $2 + 0 >/dev/null 2>&1
if test $? -eq 2
then
       echo "Il parametro $2 non e' un numero"
       exit 3
else
       if test $2 -le 0
       then
              echo "Il parametro $2 non è strettamente positivo"
              exit 3
       fi
fi

PATH=$PATH:`pwd`
export PATH

parteRicorsiva.sh $*
```

```
#!/bin/sh
# Soluzione del compito del 21 Luglio 2006
# Parte ricorsiva
# SINTASSI
# $0 dirass K
#

LISTA_FILE=

cd $1

for i in *
do
       if test -f $i -a -r $i #deve essere anche leggibile per poter contare le linee
       then
              LUNG=`wc -l <$i`
              if test `expr $2 % 2` -eq 0             # K e' pari
              then
                     if test $LUNG -eq $2
                     then

                            LISTA_FILE="$LISTA_FILE $i"
                     fi
              else                                    # K e' dispari
                     if test $LUNG -eq `expr $2 + 1`
                     then
                            LISTA_FILE="$LISTA_FILE $i"
                     fi
              fi
       fi
done

# valutazione della condizione nel direttorio corrente
if test -n "$LISTA_FILE" # se c'è almeno un file
then
       echo "Trovato direttorio `pwd`"
       echo "Invoco parte C con lista file = $LISTA_FILE"
       partec $LISTA_FILE
fi

# invocazione ricorsiva nella gerarchia
for i in *
do
    if test -d $i -a -x $i
    then
       parteRicorsiva.sh "$1/$i" $2
    fi
done
```

```
/* SOLUZIONE DEL 21 LUGLIO 2006 - PARTE C*/
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#define PERM 0644 /* in UNIX */

typedef int pipe_t[2];

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

	/* ========= VARIABILI LOCALI ============ */
	int M;                          // numero di figli/file da creare
	int i, j;                       // variabile ausiliari dei for
	int lunghezza;                  // lunghezza della linea
	int fdRead, fdCreato;      // file descriptors
	int linee_scritte;         // numero di linee scritte sul file
	pipe_t piped;              // pipe di comunicazione tra figlio e nipote
	char * FCreato;                 // nome del file da creare con concatenazione
	int contalinea;                 // contatore di linea
	char linea[512];    // buffer dove memorizzare la linea corrente letta dal file
	char ch;                        // buffer per caratteri della read
	int pid;                        // pid del processo con la fork()
	int status;                     // stato di ritorno della wait
	/* ====================================== */


	M=argc-1;

	if(M==0)
	{
		printf("Errore nel numero degli argomenti: ce ne deve essere almeno
		uno\n");
		exit(-1);
	}


	/* Ciclo di creazione dei figli */
	for(i=0; i<M; i++)
	{
		if((pid=fork())<0)
		{
			printf("Errore nella fork\n");
			exit(-2);
		}

		if(pid==0) /* CODICE DEL FIGLIO I-ESIMO */
		{
			/* Creo la socket privata tra figlio e nipote*/
			if(pipe(piped)<0)
			{
				printf("Errore di creazione della pipe\n");
				exit(-3);
			}

			/* Creo il nipote */
			if((pid=fork())<0)
			{
				printf("Errore nella fork\n");
				exit(-4);
			}

			if(pid==0) /* CODICE DEL NIPOTE*/
			{
				/* Scegliamo una apertura con I/O pointer separato tra
				figlio e nipote, per ragioni di semplicità del codice.
				Se, invece, si fosse scelta un I/O pointer condiviso si
				dovevano  sincronizzare figlio e nipote per mezzo di
				una seconda pipe oppure usando i segnali UNIX.
				Quest'ultima soluzione sarebbe più efficiente, in
				quanto riduce il numero di accessi al file system e
				quindi al disco. */
				if((fdRead = open(argv[i+1], O_RDONLY))<0)
				{
					printf("Errore nella apertura del file %s\n",
					argv[i+1]);
					exit(-7);
				}

				/* Chiudo il lato di lettura della pipe */
```

```
                close(piped[0]);

                /* Cominciamo la lettura dal file */
                contalinea=0;
                j=0;
                while(read(fdRead, &linea[j], 1)>0)
                {
                        if(linea[j] == '\n')
                        {
                                if((contalinea % 2) !=0 ) /* linea dispari */
                                /* scrivo la lunghezza della linea sulla pipe*/
                                        write(piped[1], &j, sizeof(int));
                                contalinea++;
                                j=0;
                        }
                        else
                                j++;
                }

                exit(0);
        }

        /* continua codice del figlio ...*/

        FCreato = (char *) malloc(strlen(argv[i+1]) + 5);
        /* 5 = "pari"+ terminatore */
        if(FCreato == NULL)
        {
                printf("Errore nella allocazione della memoria per il nome di
                FCreato\n");
                exit(-5);
        }

        strcpy(FCreato, argv[i+1]);
        strcat(FCreato, "pari");

        /* Tento la creazione del file Fcreato*/
        if((fdCreato=creat(FCreato, PERM))<0)
        {
                printf("Errore nella creazione del file %s\n", FCreato);
                exit(-6);
        }

        if((fdRead = open(argv[i+1], O_RDONLY))<0)
        {
                printf("Errore nella apertura del file %s\n", argv[i+1]);
                exit(-7);
        }

        /* Chiudo il lato di scrittura della pipe */
        close(piped[1]);

        /* Ciclo di lettura del figlio */
        contalinea=0;
        linee_scritte=0;
        j=0;
        while(read(fdRead, &linea[j], 1)>0)
        {
                if(linea[j] == '\n')
                {
                        if((contalinea % 2) ==0 ) /* linea pari */
                        {
                                /* leggo la lunghezza della linea sulla pipe*/
                                read(piped[0], &lunghezza, sizeof(int));

                                if(j<lunghezza)
                                {
                                        write(fdCreato, linea, j);
                                        linee_scritte++;
                                }

                        }
                        contalinea++;
                        j=0;
                }
                else
                        j++;
        }

        exit(linee_scritte);

} // FINE CODICE FIGLIO
```

```
        } // FINE FOR DELLA FORK()

        /* CODICE DEL PADRE */
        /* Aspetto i vari figli */
        while((pid=wait(&status))>0)
        {
               printf("Il figlio con PID %d è morto con valore di ritorno %d\n", pid,
               (status >> 8)&0xFF );
        }

}
```