SISTEMI OPERATIVI e
LABORATORIO DI SISTEMI OPERATIVI
(A.A. 05-06) – 22 Settembre 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 un numero variabile ***dispari*** di parametri: il primo deve essere il **nome assoluto di un direttorio** che identifica una gerarchia (**G**) all'interno del file system, mentre gli altri devono essere nomi di file (**F0... FM-1**). Il programma deve cercare nella gerarchia **G** specificata (compresa la radice) tutti i direttori che contengono gli M file **F0... FM-1**. 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**).

La parte in C accetta un numero variabile ***pari*** di parametri che rappresentano nomi di file **F0...FM-1**: quindi **M** deve essere un numero **pari** e deve essere controllato! Il processo padre deve generare **un processo figlio** (**P**) che a sua volta deve creare **un processo nipote** (**N**): il processo figlio è associato ai primi M/2 file (**F0, ... FM/2-1)**, mentre il processo nipote al resto degli M/2 file (**FM/2 ... FM-1**). La **coppia** figlio e nipote esegue concorrentemente leggendo dai file associati: i processi figlio P e nipote N devono leggere via via dai file ad essi associati iniziando da F0 il figlio e da FM/2 il nipote e terminando con FM/2-1 il figlio e da FM-1 il nipote; entrambi devono calcolare quanti caratteri alfabetici minuscoli sono presenti nel file analizzato (QP per il figlio e QN per il nipote). Quindi dopo ogni calcolo, il processo figlio P deve, per ogni file analizzato, inviare al processo nipote N una struttura contenente il nome del file analizzato dal processo figlio P e il numero QP (si consideri per semplicità intero). Il processo nipote, sempre dopo ogni calcolo, deve ricevere tale struttura e controllare se QN per il file correntemente analizzato è minore del QP ricevuto. In tal caso, deve inviare al padre la struttura ricevuta dal figlio P, altrimenti deve inviare una struttura contenente il nome del file analizzato dal processo nipote N e il numero QP.
Il padre deve ricevere tutte le strutture ricevute dal nipote N e, per ognuna, stampare su standard output il nome del file e il numero di caratteri alfabetici minuscoli in esso contenuto.

```
#!/bin/sh
# Esame di Sistemi Operativi del 22 Settembre 2006
# Parte di controlli sui parametri
# $0 dirass F0 F1 ... FM-1
#

# controllo numero di parametri
if test `expr $# % 2` -eq 0 -o $# -lt 3
then
    echo "Errore nel numero dei parametri: devo essere in numero dispari e
superiore a 2"
    exit 1
fi

# controllo primo parametro
case $1 in
/*) if test ! -d $1 -o ! -x $1
    then
      echo "Il primo parametro non e' un direttorio oppure non e' accessibile"
      exit 1
    fi;;
*)  echo "nome di direttorio non assoluto"
    exit 1;;
esac

# impostazione della variabile path
PATH=$PATH:`pwd`
export PATH

# invoco la parte ricorsiva con i nomi dei file come parametri
parteRicorsiva.sh $*
```

```
#!/bin/sh
# Esame di Sistemi Operativi del 22 Settembre 2006
# Parte ricorsiva
# $0 F0 F1 ... FM-1
#

cd $1

shift

TROVATO=true

# HP: sono gia' nel direttorio di analisi (e.g. quello radice)

for i in $*
do
    echo "Cerco file $i..."
    if test ! -f $i
    then
      echo "File $i non trovato"
      TROVATO=false
      break
    fi
done

# Attenzione: l'operatore -eq non si può applicare alle stringhe!
#
if test $TROVATO = true
then
    echo "Trovato direttorio `pwd` che soddisfa i requisiti"
    echo "Invoco la parte C con parametri $*"
    parteC $*
fi

#invocazione ricorsiva
for i in *
do
    if test -d $i -a -x $i
    then
      $0 $*
    fi
done
```

```c
/* SOLUZIONE DEL 22 SETTEMRBE 2006 - PARTE C*/
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>


typedef struct {
      char nomefile[80];
      int qty;
} MSG;

typedef int pipe_t[2];

/* FUNZIONE CHE CALCOLA IL NUMERO DI CARATTERI ALFABETICI MINUSCOLI IN UN FILE
IL CUI NOME E' PASSATO COME PARAMETRO */
int calcolaCaratteri(char * nomefile)
{
      int fd;           // file descriptor
      char ch;    // carattere buffer di lettura
      int cont=0; // numero di caratteri alfabetici minuscoli

      if((fd=open(nomefile, O_RDONLY))<0)
      {
            printf("Errore nella apertura del file %s...\n", nomefile);
            return 0;
      }

      while(read(fd, &ch, 1)>0)
      {
            if(ch >= 'a' && ch <= 'z')
                  cont++;
      }

      close(fd);

      printf("Nel file %s ho trovato %d caratteri alfabetici minuscoli\n",
      nomefile, cont);

      return cont;
}


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

      /* ========= VARIABILI LOCALI =========== */
      int M;                  // numero di file passati come parametri
      pipe_t pdPN;            // pipe tra padre e nipote
      pipe_t pdFN;            // pipe tra figlio e nipote
      MSG msg;                // struttura di scambio tra processi
      int i;                  // variabile ausiliari dei for
      int conta_struct=0;     // numero di strutture
      int num_car;            // numero di caratteri alfabetici minuscoli letti in un file
      int pid;                // pid del processo con la fork()
      int status;             // stato di ritorno della wait
      /* ====================================== */
```

```
M=argc-1;

if( (M==0) || (M%2 != 0))
{
	printf("Errore nel numero degli argomenti: devono essere pari\n");
	exit(-1);
}

/* Creo la pipe tra padre e nipote */
if(pipe(pdPN)<0)
{
	printf("Errore nella creazione della pipe padre nipote\n");
	exit(-3);
}

if((pid=fork())<0)
{
	printf("Errore nella fork del padre\n");
	exit(-2);
}

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

	if((pid=fork())<0)
	{
		printf("Errore nella fork del figlio\n");
		exit(-2);
	}

	if(pid==0) // CODICE DEL NIPOTE
	{
		close(pdPN[0]); // il nipote scrive verso il padre
		close(pdFN[1]); // il nipote legge dal figlio

		/* Inizio il ciclo delle letture */
		for(i=M/2; i<M; i++)
		{
			num_car=calcolaCaratteri(argv[i+1]);

			/* leggo la struttura dal processo figlio */
			read(pdFN[0], &msg, sizeof(msg));

			printf("NIPOTE: Il processo figlio mi ha spedito il
			nomefile %s con qty=%d caratteri alfabetici minuscoli\n",
			msg.nomefile, msg.qty);

			if(msg.qty <= num_car)
			{
				strcpy(msg.nomefile,argv[i+1]);
				msg.qty=num_car;
```

```
                        }

                        printf("NIPOTE: Spedisco al padre la struttura [%s,%d]\n",
                        msg.nomefile, msg.qty);

                        /* invio al padre la struttura ricevuta */
                        write(pdPN[1], &msg, sizeof(msg));
                  }

                  exit(0);

            } // FINE CODICE NIPOTE

            /* Il figlio non utilizza la pipe PN*/
            close(pdPN[0]); close(pdPN[1]);

            /* ...ed utilizza il lato scrittura della pipe FN*/
            close(pdFN[0]);

            /* Inizio il ciclo delle letture */
            for(i=0; i<M/2; i++)
            {
                  num_car=calcolaCaratteri(argv[i+1]);
                  msg.qty=num_car;
                  strcpy(msg.nomefile,argv[i+1]);

                  printf("FIGLIO: Spedisco al nipote la struttura[%s,%d]\n",
                  msg.nomefile, msg.qty);

                  /* invio al nipote la struttura */
                  if(write(pdFN[1], &msg, sizeof(msg))==sizeof(msg))
                        conta_struct++;
            }

            exit(0);

      } // FINE CODICE FIGLIO

      /* CODICE DEL PADRE */

      /* Chiusura lati non utilizzati*/
      close(pdPN[1]);

      for(i=0; i<M/2; i++)
      {
            read(pdPN[0], &msg, sizeof(msg));
            printf("Padre ha letto struttura con: qty=%d e nomefile=%s\n", msg.qty,
            msg.nomefile);
      }
}
```