SISTEMI OPERATIVI e
LABORATORIO DI SISTEMI OPERATIVI
(A.A. 05-06) – 23 Giugno 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, multiplo intero di 4 e strettamente minore di **11**. Il programma deve cercare nella gerarchia **G** specificata tutti i direttori che contengono un numero di file esattamente uguale a **K**. Si riporti il nome assoluto di tali direttori sullo standard output. In ogni direttorio trovato, si deve invocare la parte in C **due volte:** la prima volta, passando come parametri i file di posizione dispari (F1, F3, ...FK-1) e la seconda volta, i file di posizione pari (F2, F4, ...FK).

La parte in C accetta un numero variabile pari di parametri (si effettui il necessario controllo) che rappresentano nomi di file **F0...FM-1**. Il processo padre deve generare **M processi figli** (**P0** … **PM-1**): ogni processo figlio è associato ad uno dei file **Fi**. I figli si devono considerare a coppie ordinate: ogni coppia è costituita dal processo di indice pari Pp (i = 0, 2, ... M-2) e dal processo di indice dispari Pd (i = 1, 3, ... M-1). La comunicazione in ognuna di tali coppie deve essere dal processo Pp al processo Pd; la prima azione che dovrà compiere il processo Pd sarà quella di creare un file (**Fcreato**) il cui nome sia la concatenazione del nome del file associato **Fi** con la stringa ".dispari". Ogni processo figlio esegue concorrentemente leggendo i caratteri del file ad esso associato **Fi:** dopo la lettura di ogni carattere (Cp) da parte del processo Pp, questo viene comunicato al processo Pd, il quale lo riceve dopo aver letto il proprio carattere (Cd); il processo Pd deve confrontare il proprio carattere Cd con il carattere ricevuto Cp e se Cd risulta strettamente maggiore di Cp lo scrive sul file **Fcreato. Si può supporre che i file letti dalla coppia di processi abbiano lunghezza uguale!** Ogni processo figlio deve ritornare al padre il numero di caratteri letti dal proprio file. 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 del compito del 23 Giugno 2006
# File di controllo parametri
# SINTASSI:
# $0 dirass K
#

# Controllo il numero dei parametri
case $# in
2) ;;
*) echo "Errore nel numero dei parametri"
   exit 1;;
esac

# controllo primo parametro direttorio assoluto
case $1 in
/*) if test ! -d $1 -o ! -x $1
    then
      echo "Errore nel primo parametro: non direttorio o non accessibile"
      exit 2
    fi;;
*) echo "Errore nel primo parametro: non e' assoluto"
   exit 2;;
esac


# controllo sul parametro K

expr $2 + 0 >/dev/null 2>&1
if test $? -ne 2 # allora e' un numero
then
    if test $2 -le 0 -o `expr $2 % 4` -ne 0 -o $2 -ge 11
    then
      echo "Il numero $2 non e' corretto"
      exit 3
    fi
fi

PATH=$PATH:`pwd`
export PATH

parteRicorsiva.sh $*
```

```
#!/bin/sh
# Soluzione del compito del 23 Giugno 2006
# Parte ricorsiva
# SINTASSI
# $0 dirass K
#

CONTAFILE=0
FILE_PARI=
FILE_DISPARI=

cd $1


# controllo sul direttorio corrente (compresa la radice della gerarchia)
for i in *
do
    if test -f $i
    then
      CONTAFILE=`expr $CONTAFILE + 1`
      if test `expr $CONTAFILE % 2` -eq 0
      then
          FILE_PARI="$FILE_PARI $i"
      else
          FILE_DISPARI="$FILE_DISPARI $i"
      fi
    fi
done


# controllo sui file trovati e invocazione della parte C
if test $CONTAFILE -eq $2
then
    echo "Trovato direttorio $1"
    echo "Invoco la parte C la prima volta con parametri \"$FILE_DISPARI\""
    parteC $FILE_DISPARI
    echo "Invoco la parte C la seconda volta con parametri \"$FILE_PARI\""
    parteC $FILE_PARI
fi

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

```c
/* SOLUZIONE DEL 23 GIUGNO 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[])
{
	int M;/* numero di file della riga di comando = numero di figli da creare */
	pipe_t *piped;/* array di pipe per la comunicazione tra coppie di processi */
	int i,j;		/* variabile per cicli for */
	int pid;		/* pid del figlio creato */
	char *Fcreato;		/* nome del file concatenato con Fi*/
	int fdCreato;		/* file descriptor di Fcreato */
	int fdRead;		/* file descriptor del file Fi */
	int conta_car=0;  /* contatore di caratteri letti dal file Fi*/
	char ch, chd;		/* carattere letto dal file Fi */
	int status;		/* variabile di status della wait */


	M=argc-1;

	/* Controllo il numero dei parametri che è variabile */
	if(M<2)
	{
		printf("Errore nel numero degli argomenti: devono essere almeno 2\n");
		exit(-1);
	}

	/* Verifico che il numero dei parametri*/
	if((M % 2)!=0)
	{
		printf("Errore nel numero degli argomenti: devono essere pari\n");
		exit(-2);
	}

	/* Essendo la comunicazione a coppie Pp/Pd, devo creare M/2 pipes*/
	piped = (pipe_t *) malloc((M/2)*sizeof(pipe_t));
	if(piped==NULL)
	{
		printf("Errore nella allocazione della memoria per le pipes\n");
		exit(-3);
	}

	for(i=0; i<M/2; i++)
	{
		if(pipe(piped[i])<0)
		{
			printf("Errore di creazione della pipe\n");
			exit(-4);
		}
	}

	/* Adesso creo gli M figli associati ai file */
```

```
for(i=0; i<M; i++)
{
	if((pid=fork())<0)
	{
		printf("Errore nella fork\n");
		exit(-5);
	}

	if(pid==0) /* CODICE DEL FIGLIO I-ESIMO */
	{

	/*Ogni processo figlio apre in lettura il proprio file associato*/
		if((fdRead = open(argv[i+1], O_RDONLY))<0)
		{
			printf("Errore nella apertura del file %s\n", argv[i+1]);
			exit(-6);
		}

		if((i%2)==0) /* PROCESSO PARI */
		{
			/* Ogni figlio chiude le pipe che non usa */
			for(j=0; j<M/2; j++)
			{
				close(piped[j][0]);
				if(j!=i/2) close(piped[j][1]);
			}

			/* ciclo di lettura dal file Fi*/
			while(read(fdRead, &ch, 1)>0)
			{
				write(piped[i/2][1], &ch, 1);
				conta_car++;
			}
		}
		else /* PROCESSO DISPARI */
		{
			/* Ogni figlio chiude le pipe che non usa */
			for(j=0; j<M/2; j++)
			{
				close(piped[j][1]);
				if(j!=((i-1)/2)) close(piped[j][0]);
			}

			/* Concateno le stringhe */
			Fcreato = (char *) malloc(strlen(argv[i+1]) + 9); /* 9 =
			".dispari"+ terminatore */
			if(Fcreato == NULL)
			{
				printf("Errore nella allocazione della memoria per il nome di
				FCreato\n");
				exit(-7);
			}

			strcpy(Fcreato, argv[i+1]);
			strcat(Fcreato, ".dispari");

			/* Tento la creazione del file Fcreato*/
			if((fdCreato=creat(Fcreato, PERM))<0)
```

```
                        {
                              printf("Errore nella creazione del file %s\n",
                              Fcreato);
                              exit(-8);
                        }

                        /* ciclo di lettura dal file Fi*/
                        while(read(fdRead, &chd, 1)>0)
                        {
                              read(piped[(i-1)/2][0], &ch, 1);
                              if(chd>ch)
                                    write(fdCreato, &chd, 1);
                              conta_car++;
                        }

                  } /* FINE FIGLIO DISPARI */

                  exit(conta_car);
            } /* FINE DEL FIGLIO I-ESIMO*/
      }

      /* CODICE DEL PADRE*/

      /* chiude tutte le pipe */
      for(i=0; i<M/2; i++)
      {
            close(piped[i][0]);
            close(piped[i][1]);
      }

      /* 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 );
      }

}
```