SISTEMI OPERATIVI e
LABORATORIO DI SISTEMI OPERATIVI
(A.A. 06-07) – 30 MARZO 2007

IMPORTANTE:

1) Fare il login sui sistemi in modalità Linux usando il proprio **username** e **password**.
2) I file prodotti devono essere collocati in un **sottodirettorio** della propria **HOME** directory che deve essere creato e avere nome **ESAME30Mar07-1-1.** FARE ATTENZIONE AL NOME DEL DIRETTORIO, in particolare alle maiuscole e ai trattini indicati. Verrà penalizzata l'assenza del direttorio con il nome indicato e/o l'assenza dei file nel direttorio specificato, al momento della copia automatica del direttorio e dei file. **ALLA SCADENZA DEL TEMPO A DISPOSIZIONE VERRÀ INFATTI ATTIVATA UNA PROCEDURA AUTOMATICA DI COPIA, PER OGNI STUDENTE DEL TURNO, DEI FILE CONTENUTI NEL DIRETTORIO SPECIFICATO.**
3) Il tempo a disposizione per la prova è di **120 MINUTI** per lo svolgimento di tutto il compito e di **75 minuti** per lo svolgimento della sola parte C.
4) Non è ammesso **nessun tipo di scambio di informazioni** né verbale né elettronico, pena la invalidazione della verifica.
5) L'assenza di commenti significativi verrà penalizzata.
6) **AL TERMINE DELLA PROVA È INDISPENSABILE CONSEGNARE IL TESTO DEL COMPITO (ANCHE IN CASO CHE UNO STUDENTE SI RITIRI): IN CASO CONTRARIO, NON POTRÀ ESSERE EFFETTUATA LA CORREZIONE DEL COMPITO MANCANDO IL TESTO DI RIFERIMENTO.**

Esercizio

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

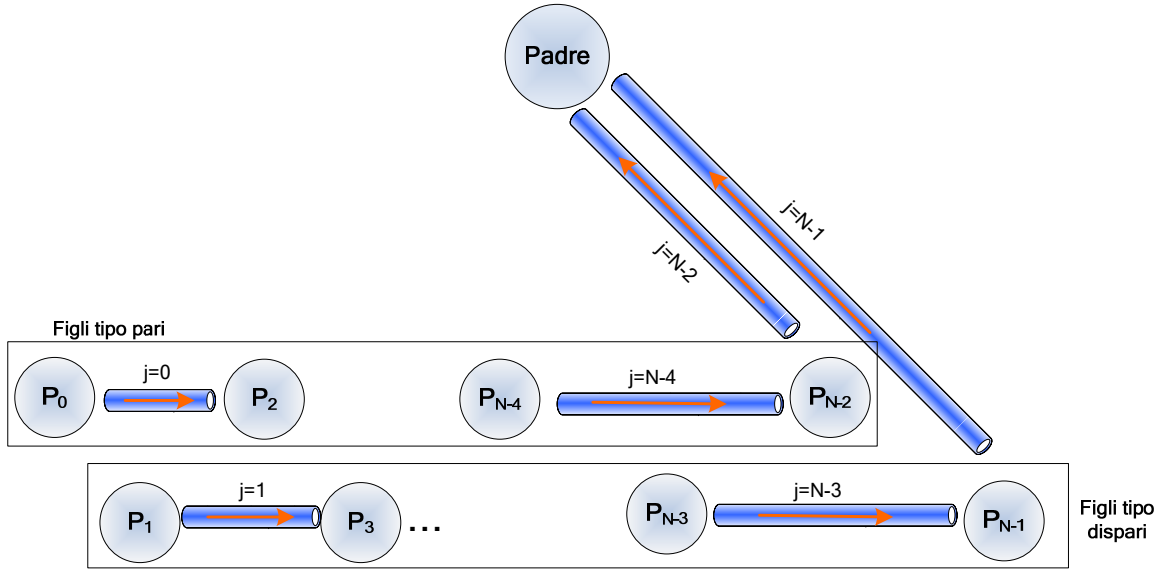
La parte in Shell deve prevedere due parametri: il primo deve essere il **nome assoluto di un direttorio** che identifica una gerarchia (**G**) all'interno del file system, mentre il secondo parametro deve essere considerato un singolo carattere **(C)**. Il programma deve cercare nella gerarchia **G** specificata tutti i direttori (compresa la radice) che contengono almeno un file il cui nome sia di 4 caratteri e i cui caratteri pari, sempre del nome, siano tutti uguali al carattere **C**. Si riporti il nome assoluto di tali direttori sullo standard output. Al termine dell'intera esplorazione ricorsiva di **G**, se il numero di file trovati (cioè che soddisfano la condizione precedente) è pari allora si deve invocare la parte in C, passando come parametri i nomi assoluti dei file trovati (**F1, F2, ... FN**) e il carattere **C**.

La parte in C accetta un numero variabile di parametri che rappresentano nomi assoluti di file (**F1, F2, ... FN**) e un singolo carattere (**C**): il numero di file è variabile e comunque superiore ad 1, ma pari (si effettui il necessario controllo su tale numero).
Il processo padre deve creare N processi figli (P0 … PN-1): ogni processo figlio è associato ad uno dei file Fi. I figli vengono distinti in base al loro indice in due tipi: *figli pari* e *figli dispari*. Ogni processo figlio pari deve cercare le occorrenze del carattere **C** nei caratteri di posizione pari del file associato Fi; mentre ogni processo figlio dispari deve cercare le occorrenze del carattere **C** nei caratteri di posizione dispari del file associato Fi.
Ognuno dei figli (sia pari che dispari), ad eccezione dei primi processi di ognuno dei due tipi, una volta calcolato il suo numero di occorrenze deve ricevere il conteggio accumulato dal figlio dello stesso tipo (cioè pari o dispari) precedente e, una volta sommato al proprio conteggio, comunicarlo al figlio successivo dello stesso tipo; l'ultimo processo di ognuno dei due tipi deve comunicare il conteggio di occorrenze totale (cioè le sue più quelle ricevute) al padre. Ogni processo figlio deve ritornare al padre il numero di occorrenze trovate nel proprio file associato.
Il padre deve ricevere dai due ultimi figli (prima quello pari e poi quello dispari) i due conteggi totali e li deve stampare su standard output. Quindi, dopo che i figli sono terminati, il padre deve stampare su standard output i PID di ogni figlio con il corrispondente valore ritornato.

```
#!/bin/sh
#
# Soluzione dell'appello di SONOD del 30 Marzo 2007
# File dei controlli
#
# $0 dirass C

# controllo numero dei parametri
case $# in
2);;
*)
      echo "Uso: $0 dirass C..."
      exit 1;;
esac

# controllo primo parametro direttorio assoluto
case $1 in
/*)
      if test ! -d $1 -o ! -x $1
      then
          echo "Direttorio inesistente o inaccessibile"
          exit 2
      fi;;
*)
      echo "Nome non assoluto"
      exit 2;;
esac

# controllo secondo parametro
case $2 in
?);;
*)
      echo "Secondo parametro non e' singolo carattere"
      exit 3;;
esac

# impostazione di PATH
export PATH=$PATH:`pwd`
export TMP_FILE="/tmp/files$$"

# invocazione della parte ricorsiva
ricorsione.sh $*

# conto le linee del file
LINEE=`wc -l <$TMP_FILE`
if test `expr $LINEE % 2` -eq 0 # se il numero di file trovati e' pari
then
  # invocazione della parte c
    partec `cat $TMP_FILE` $2
fi
```

```
#!/bin/sh
#
# Soluzione dell'esame del 30 Marzo 2007
# Parte ricorsiva
#
# $0 dirass C

# entro nel direttorio di analisi
cd $1

C=$2
trovato_file=false

# esploro il contenuto del direttorio (compresa la radice)
for i in *
do
  if test -f $i
  then
      case $i in
      $C?$C?)
            # $i contiene il nome relativo semplice di un file che soddisfa le
            # condizioni
            trovato_file=true;
            echo "$1/$i" >> $TMP_FILE
      esac
  fi
done

if test $trovato_file = true
then
    echo "Trovato direttorio $1 che soddisfa le condizioni..."
fi

for i in *
do
  if test -d $i -a -x $i
  then
      $0 "$1/$1" $2
  fi
done
```

IMPORTANTE: Si richiede di disegnare (in maniera chiara!) lo schema di comunicazione scelto per risolvere la parte C (ad es., pipe con le direzioni di comunicazione e gli indici).



✍ Quante pipe sono state utilizzate?

N = _argc-2___

✍ Perché?

Ogni figlio scrive sulla pipe col suo stesso indice (j=i) e legge dalla pipe con indice j=i-2. Questo, ad eccezione dei primi due figli (i=0 ed i=1), i quali non leggono da alcuna pipe. Infine, gli ultimi due figli (i=N-2 e i=N-1) scrivono i loro conteggi verso il padre. Quindi, si sono scelte esattamente N pipe, tante quanti sono i figli.

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <string.h>

#define PERM 0644 /* in UNIX */

typedef int pipe_t[2];

int main(int argc, char **argv)
{
	/* Local variables */
	int N;			/* numero di argomenti come file */
	int i, j;		/* variabile ausiliaria per i loop */
	int pid;		/* process identifier di supporto */
	int status;		/* variabile per lo stato della wait */
	int Fi;			/* file descriptor per ognuno dei file passati */
	pipe_t *piped;	/* descrittori delle N pipe */
	int occ;		/* contatore delle occorrenze di C*/
	int occ_prev;	/* occorrenze del figlio precedente */
	char C;			/* carattere argv[argc-1][0] salvato per comodita'  */
	char ch;		/* buffer per la lettura da file */
	/* -----------------*/

	/* Controllo sul numero dei parametri */
	N=argc-2; /* Tolgo argv[0] e argv[argc-1]*/
	if( (N < 2) || ( N%2 != 0))
	{
		printf("Errore nel numero dei parametri. Uso: %s f1 f2 ... fN C
		\n",argv[0]);
		exit(-1);
	}

	C=argv[argc-1][0];

	/* Creazione delle pipe per la comunicazioni tra i figli e alla fine con il
	padre */
	piped = (pipe_t *) malloc(N*sizeof(pipe_t));
	if(piped==0)
	{
		printf("Errore nella malloc\n");
		exit(-2);
	}

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

	/* ciclo di creazione degli N figli */
	for (i=0; i<N; i++)
	{
		if((pid=fork())<0)
		{
			printf("Errore nella creazione di un figlio\n");
			exit(-2);
		}
```

```
	else if(pid == 0) /*codice del figlio*/
		break; // esco dal ciclo
} // file ciclo for


if(pid==0) /*codice del figlio*/
{
	/* Chiudo lati pipe che non utilizzo*/
	/* Figlio i scrive su pipe i e legge da pipe i-2 (se i>1)*/
	for (j=0; j<N; j++)
	{
		if(j!=i)
			close(piped[j][1]);
		if(i<2) /* primi due figli di entrambi i tipi */
			close(piped[j][0]);
		else
			if(j!=i-2)
				close(piped[j][0]);
	}

	printf("Sono figlio con PID %d e apro il mio file %s\n", getpid(),
	argv[i+1]);

	/* Apro il file associato "Fi" per leggere */
	if((Fi=open(argv[i+1], O_RDONLY))<0)
	{
		printf("Errore nella apertura del file \"%s\"\n", argv[i+1]);
		exit(-5);
	}

	if(i%2!=0) /*figlio di tipo dispari --> fai prima lettura a vuoto */
		read(Fi, &ch, 1);

	occ=0;

	while(read(Fi, &ch, 1)>0) /* qui i due tipi di figli sono allineati al
	loro primo carattere utile*/
	{
		if (ch == C)
			occ++;
		/* Altra lettura a vuoto per saltare il prossimo carattere */
		read(Fi, &ch, 1);
	}

	/* occ contiene ora il numero di occorrenze trovate fino all'EOF*/
	if(i>1) /* ad esclusione dei primi due figli */
	{
		/* aspetto il conteggio dal mio fratello precedente */
		read(piped[i-2][0], &occ_prev, sizeof(int));

	}
	else
		occ_prev=0;

	/* ci sommo le mie occorrenze ...*/
	occ_prev+=occ;

	/*... e spedisco al successivo (al padre se sono ultimo o penultimo)*/
	write(piped[i][1], &occ_prev, sizeof(int));

	/* ritorno le occorrenze mie al padre */
```

```
            exit(occ);
      }

      /* codice del padre */

      /* Chiude le pipe che non utilizza */
      for(i=0; i<N; i++)
      {
            close(piped[i][1]);
            if(N-i<2) /* lascio aperti i lati lettura dei due ultimi figli */
                  close(piped[i][0]);
      }

      for(i=N-1; i>=N-2; i--)
      {
            read(piped[i][0], &occ, sizeof(int));
            printf(" Figlio di indice %d ha tornato conteggio totale %d\n", i,
            occ);
      }

      for (i=0; i<N; i++)
      {
            pid=wait(&status);
            printf("Il figlio con PID %d ï¿½ritornato con valore %d\n", pid,
            (status>>8)&0xFF);
      }
}
```