# SISTEMI OPERATIVI e
# LABORATORIO DI SISTEMI OPERATIVI
# (A.A. 02-03) - 19 MARZO 2003

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 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 i **nomi relativi (semplici) di file** (**F1, F2, ..., FN**). Il programma deve cercare nella gerarchia **G** specificata tutti i direttori il cui **nome contiene almeno un carattere alfabetico maiuscolo** e **che contengono tutti i file con nome F1, F2, ..., FN**: si riporti il nome assoluto di tali direttori sullo standard output. Per ogni direttorio trovato, si deve invocare la parte C passando come parametri i nomi dei file F1, F2, ..., FN.

La parte in C accetta un numero variabile di parametri che rappresentano nomi relativi semplici di file f0, f1, f2 ... fN-1 . Il processo padre deve generare un numero di **processi figli** pari al numero di file passati come parametri: ogni processo figlio è associato ad uno dei file **fi** (**con i che varia da 0 a N-1**). Ognuno di tali processi figli esegue concorrentemente leggendo dal file associato una selezione dei caratteri riportandoli sullo standard output. I processi figli devono coordinarsi vicendevolmente in modo che le scritture su standard output avvengano in uno specifico ordine. In dettaglio, ogni processo figlio, ciclicamente partendo dal primo processo fino all'ultimo processo, deve selezionare i caratteri di indice i + k * N (**con i che varia da 0 a N-1** e con **k che varia da 0, 1, 2, ...**) e deve stampare il carattere selezionato solo se il processo associato al file di indice precedente ha già effettuato la sua stampa. Ad esempio supponendo che i parametri di invocazione siano 3 con nomi file0, file1 e file2, il processo di indice 0 deve leggere da file0 il carattere di indice 0 (=0+0*3) e riportarlo su standard output, quindi il processo di indice 1 deve leggere da file1 il carattere di indice 1 (=1+0*3) e riportarlo su standard output (dopo essersi coordinato con il processo precedente), infine il processo di indice 2 deve leggere da file2 il carattere di indice 2 (=2+0*3) riportarlo su standard output (dopo essersi coordinato con il processo precedente); a questo punto si deve ricominciare nel senso che il processo di indice 0 deve leggere da file0 il carattere di indice 3 (=0+1*3) e riportarlo su standard output (dopo essersi coordinato con il processo precedente), quindi il processo di indice 1 deve leggere da file1 il carattere di indice 4 (=1+1*3) e riportarlo su standard output (dopo essersi coordinato con il processo precedente), infine il processo di indice 2 deve leggere da file2 il carattere di indice 5 (=2+1*3) riportarlo su standard output (dopo essersi coordinato con il processo precedente); etc..
Si consideri comunque che le lunghezze dei file siano tali che i processi terminano in cascata, dal primo all'ultimo.

```
# file es190303
# soluzione del compito del 19 marzo 2003
# parte principale shell

# controllo del numero dei parametri
if test $# -lt 2
then
  echo Uso: $0 dirassoluto F1 .. FN
  exit
fi

# controllo sul primo parametro
case $1 in
/*) if test ! -d $1 -o ! -r $1 -o ! -x $1
    then
      echo $1 non esistente o non accessibile
      exit
    fi ;;
*) echo $1 non e"'" assoluto
   exit ;;
esac

# controllo sugli altri parametri
for i in $*
do
  if test $i != $1
  then
    case $i in
    */*) echo $i non relativo
         exit ;;
    esac
  fi
done

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

# invocazione della parte ricorsiva
cerca190303 $*
```

```
# file cerca190303
# soluzione del compito del 19 marzo 2003
# parte ricorsiva shell

# dice se il nome del direttorio c'e' un carattere maiuscolo
trovato=false
# conta i file trovati
conta=0

# attenzione: nella radice il nome e' assoluto
case $1 in
 *[A-Z]*) trovato=true ;;
esac

cd $1
# scartiamo il primo parametro
shift

if test $trovato = true
then
  for i in $*
  do
    if test -f $i
    then
      conta=`expr $conta + 1`
    fi
  done
  if test $conta -eq $#
  then
    echo Trovato direttorio `pwd`
    partec190303 $*
  fi
fi

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

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

main(int argc, char *argv[])
{
  typedef int pipe_t[2];
  pipe_t *p;

  int N,lettura,scrittura,ncar,k,i,j,fdr,pid;
  char c,ok;

  if (argc < 2)
  {
    printf("Errore nel numero dei parametri\n");
    exit(1);
  }

  /* numero di file */
  N = argc-1;

  /* alloco lo spazio per le n pipe */
  if ((p = (pipe_t *)malloc(N * sizeof(pipe_t)))==0)
  {
    printf("errore nella malloc\n");
    exit(1);
  }

  /* creazione delle pipe */
  for (i=0; i<N; i++)
    if(pipe(p[i])<0)
    {
      printf("errore nella creazione della pipe\n");
      exit(1);
    }

  /* creazione dei figli*/
  for (i=0; i<N; i++)
  {
    if ((pid = fork()) < 0)
    { /* fork fallita */
      printf("Errore nella fork\n");
      exit(1);
    }
```

```
  if(pid == 0)  /* figlio */
  {
    printf("figlio %d con pid %d\n", i, getpid());
    /* il figlio i usa:
       p[i][0] per leggere l'ok
       p[(i+1)%N][1] per scrivere l'ok
    */
    lettura = i;
    scrittura = (i+1) % N;
    /* chiude il lato delle pipe che non usa */
    close(p[lettura][1]);
    close(p[scrittura][0]);
    for(j=0; j<N; j++)
      if((j != lettura) && (j != scrittura))
      {
        close(p[j][0]);
        close(p[j][1]);
      }

    /* apre in sola lettura il file argv[i+1] */
    if ((fdr = open(argv[i+1], O_RDONLY)) < 0)
    {
      printf("Errore in apertura del file %s\n", argv[i+1]);
      exit(2);
    }

    ncar = 0;
    k = 0;
    while(read(fdr, &c, 1) > 0)
    {
      if(ncar == (i + k * N))
      {
        read(p[lettura][0],&ok,1);
        write (1, &c, 1);
        write(p[scrittura][1], &ok, 1)
        k++;
      }
      ncar++;
    }
    exit(0);
  }
} /* fine for */
```

```
  /* padre */

  /* da' l'ok al primo figlio */
  write(p[0][1],&ok,1);
  /* chiude i lati delle pipe che non usa */
  for(j=0; j<N; j++)
  {
    close(p[j][0]);
    close(p[j][1]);
  }

  /* aspetta tutti i figli */
  while(wait((int*)0) > 0)
  ;
}
```

**2ª versione: non teniamo conto del fatto che i processi termino in cascata**

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

main(int argc, char *argv[])
{
  typedef int pipe_t[2];
  pipe_t *p;

  int N,lettura,scrittura,ncar,k,i,j,fdr,pid, sync_ok, sync = 1;
  char c,ok;

  if (argc < 2)
  { printf("Errore nel numero dei parametri\n");
    exit(1);
  }

  /* numero di file */
  N = argc-1;

  /* alloco lo spazio per le n pipe */
  if ((p = (pipe_t *)malloc(N * sizeof(pipe_t)))==0)
  {
    printf("errore nella malloc\n");
    exit(1);
  }

  /* creazione delle pipe */
  for (i=0; i<N; i++)
    if(pipe(p[i])<0)
    {
      printf("errore nella creazione della pipe\n");
      exit(1);
    }

  /* creazione dei figli*/
  for (i=0; i<N; i++)
  {
    if ((pid = fork()) < 0)
    { /* fork fallita */
      printf("Errore nella fork\n");
      exit(1);
    }
```

```
if(pid == 0)  /* figlio */
{
  printf("figlio %d con pid %d\n", i, getpid());
  /* il figlio i usa:
     p[i][0] per leggere l'ok
     p[(i+1)%N][1] per scrivere l'ok
  */
  lettura = i;
  scrittura = (i+1) % N;
  /* chiude il lato delle pipe che non usa */
  close(p[lettura][1]);
  close(p[scrittura][0]);
  for(j=0; j<N; j++)
    if (j != scrittura)
      close(p[j][1]);

  /* apre in sola lettura il file argv[i+1] */
  if ((fdr = open(argv[i+1], O_RDONLY)) < 0)
  {
    printf("Errore in apertura del file %s\n", argv[i+1]);
    exit(2);
  }

  ncar = 0;
  k = 0;
  while(read(fdr, &c, 1) > 0)
  {
    if(ncar == (i + k * N))
    {
  /* sincronizzazione */
  /* si parte dalla pipe di lettura e si decrementa fino
     a trovare una pipe di un processo ancora attivo */
  sync_ok = 0;
  /* sync dice se ci sono altri processi
     con cui sincronizzarsi */
  /* sync_ok dice se il processo si e' sincronizzato */
  while ((sync == 1) && (sync_ok == 0))
  {
    /* se il figlio precedente NON e' terminato */
    if (read(p[lettura][0], &ok, 1) == 1)
    {
      /* mi sono sincronizzato, esco dal while */
      sync_ok = 1;
    }
```

```
        else
        {
          /* provo con il figlio precedente */
          /* se sono arrivato alla pipe di indice 0 */
          if (lettura == 0)
            /* riparto da N-1 */
            lettura = N - 1;
          else
            /* altrimenti decremento l'indice */
            lettura--;
          /* se ho fatto un giro intero,
             gli altri figli sono tutti terminati
             non ho piu' bisogno di sincronizzarmi  */
          if (lettura == scrittura)
            sync = 0;
        }
      }

          /* scrive il carattere su standard output */
          write (1, &c, 1);
          /* da' l'ok al figlio successivo */
          write(p[scrittura][1], &ok, 1);
          k++;
        }
        ncar++;
      }
      exit(0);
    }
  } /* fine for */

  /* padre */
  /* da' l'ok al primo figlio */
  write(p[0][1],&ok,1);
  /* chiude i lati delle pipe che non usa */
  for(j=0; j<N; j++)
  {
    close(p[j][0]);
    close(p[j][1]);
  }

  /* aspetta tutti i figli */
  printf("Padre aspetta i figli\n");
  while(wait((int*)0) > 0)
  ;
}
```