SISTEMI OPERATIVI e
LABORATORIO DI SISTEMI OPERATIVI
(A.A. 04-05) – 18 MARZO 2005

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 **ESAME18Mar-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 **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** (già realizzata) e una parte in **C**.

Parte in Linguaggio C

La parte in C accetta tre parametri che rappresentano due nomi di file **F1 e F2** e un numero intero positivo **M** che rappresenta il numero di linee del file **F1**. Il processo padre, dopo aver creato il file **F2**, deve generare **M processi figli** (**P1** … **PM**): ogni processo figlio **Pj** è associato alla linea j-esima del file **F1**. Ognuno di tali processi figli esegue concorrentemente, legge la linea assegnata e seleziona **il primo** carattere **Cj** della propria linea. Dopo tale selezione, ogni figlio Pj deve comunicare una informazione al figlio seguente Pj+1, a parte l'ultimo figlio che la comunica al padre: l'informazione da comunicare è costituita da un numero e da un carattere . In particolare, il primo processo figlio **P1**, dopo aver operato la propria selezione sulla linea numero 1, comunica l'informazione costituita dal numero 1 e dal carattere C1 al processo P2; il processo **P2**, dopo aver operato la propria selezione sulla linea numero 2, riceve l'informazione comunicata da P1 e passa al processo P3 l'informazione costituita dal numero 2 e dal carattere C2 se il proprio carattere è maggiore di quello ricevuto altrimenti l'informazione ricevuta da P1 e così via fino a che l'ultimo processo **PM**, dopo aver operato la propria selezione sulla linea numero M, riceve l'informazione comunicata da PM-1 e passa al processo **padre** l'informazione costituita dal numero M e dal carattere CM se il proprio carattere è maggiore di quello ricevuto altrimenti l'informazione ricevuta da PM-2. Inoltre, ogni processo Pj **scrive sul file F2** la propria linea con i caratteri invertiti cioè dall'ultimo al primo e quindi terminando la linea ('\n'); infine, deve ritornare al padre il numero di caratteri della propria linea.
Il padre ha il compito di stampare su standard output le informazioni ricevute dal processo PM e quindi, dopo che i figli sono terminati, deve stampare su standard output i PID di ogni figlio con il corrispondente valore ritornato.

```c
/* Soluzione parte C:

      i figli devono comunicare all'indietro (il figlio i-esimo comunica a quello
      i-1). Lo schema di sincronizzazione prevede che ogni figlio legga dalla pipe
      i+1 e scriva su quella i-esima. In questo modo il primo figlio (i=0) scrive
      sulla pipe 0, l'ultimo figlio (i=n-1) scrive su quella n-1 ecc. L'ultimo
      figlio puo' partire da solo (non aspetta nessuno).
      Attenzione: per girare il buffer occorre prima memorizzare il buffer invertito
      e poi scrivere tale buffer con una write sola.
*/

#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <stdio.h>

/* definisco un tipo per un array di due interi */
typedef int pipe_t[2];

/* Siccome occorre passare una duplice informazione (numero di linea e
carattere letto), si utilizza una struttura per passare in modo atomico
(cioe' senza due write/read) i dati sulle pipe */
typedef struct {
      int carattere;
      int linea;
} messaggio;


/* Inizio del programma vero e proprio */
/*
  argv[0] <=> nome programma
  argv[1] <=> F1
  argv[2] <=> F2
  argv[3] <=> M

  quindi argc vale almeno 4
*/

int main(int argc, char **argv)
{
      /* ============ VARIABILI LOCALI ============= */
      int i;            /* Contatore per la generazione delle pipe e dei figli */
      int j;            /* Contatore di servizio */
      pipe_t *p;        /* Puntatore ad un array di pipe */
      int status;       /*utilizzato come status di ritorno dei figli*/
      int pid;          /* valore di ritorno della fork o pid del processo corrente */
      int M;            /* numero di processi figli da creare */

      messaggio current, received;        /* strutture dati utilizzate per il calcolo */
      pipe_t *pipes;                      /* vettore delle pipe */
      int fdr, fdw;                       /* file descriptor */
      char buffer[256], buffer_inv[256];  /* buffer usati per contenere la linea */
      int linea,index;                    /* contatori per il ciclo della linea */
      /* ========================================== */


    /* controllo numero argomenti */
    if( argc != 4 ){
      printf("\nATTENZIONE: devi usare %s F1 F2 N",argv[0]);
      exit(-1);
    }

    /* calcolo il numero di figli da creare */
    M = atoi(argv[3]);

    /* controllo che il valore sia positivo e non zero */
    if( M <= 0 ){
```

```
  printf("\nSpecificare un numero di linee positivo, non %d",M);
}

/* Il padre apre il file di scrittura in modo da far ereditare il file
   descriptor a tutti i figli, cosi' l'I/O pointer sara' condiviso.
*/
if( (fdw = creat(argv[2],644)) < 0 ){
  printf("\nNon riesco ad aprire il file %s in scrittura",argv[2]);
  exit(-2);
}

/* allocazione del vettore delle pipe */
pipes = (pipe_t*)malloc(M * sizeof(pipe_t));
if( pipes == NULL ){
  printf("\nImpossibile allocare %d pipe",M);
  exit(-3);
}

  /* ciclo di creazione delle M pipe per i figli*/
for(i=0; i<M; i++){
  if( pipe(pipes[i]) < 0 ){
              printf("\nImpossibile creare la pipe %d-esima",i);
              exit(4);
        }
}

/* Ora il padre puo' generare gli M figli */
printf("\nPadre %d genera i figli",getpid());

for(i=0; i<M; i++){

        pid = fork();
        if( pid < 0 ){
              printf("\nImpossibile generare il %d-esimo figlio",i);
              exit(5);
        }

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

              /* chiusura delle pipe */
              for(j=0; j<M; j++){
                if( j!=i /*pipe in cui scrivo*/ && j!=(i+1)%M /*pipe da cui leggo*/){
                    close(pipes[j][0]);
                    close(pipes[j][1]);
                }
  }

  /* leggo dalla pipe i+1-esima e scrivo sulla pipe i-esima ,
  chiudo i lati non usati */
  close(pipes[i][0]);
  close(pipes[(i+1)%M][1]);

  /* apre in lettura il file passato come secondo argomento */
  if( (fdr = open(argv[1],O_RDONLY)) < 0 ){
    printf("\nFiglio PID=%d non riesce ad aprire il file %s",getpid(),argv[1]);
    exit(6);
  }

  /* il figlio i-esimo deve leggere la linea i-esima, quindi
     leggo un carattere alla volta contando tutte le linee */
  linea=0;
  index=0;
  while( read(fdr,&buffer[index],1) )
  {
        /* siamo a fine linea ? */
        if( buffer[index] == '\n' )
        {
              if( i != linea )
```

```
            {
              /* non eravamo sulla linea giusta, incrementiamo
                  il contatore di linea e riazzeriamo l'indice del
                  buffer */
              linea++;
              index = 0;
            }
            else
                  if( i == linea )
                  {
                        /* eravamo sulla linea giusta, ora l'ho letta tutta */
                        /* ATTENZIONE: se la mia linea e' costituita dal solo '\n'
                           devo riportare quello come ultimo carattere della linea
                        */
                        if( index > 0 )
                          current.carattere = buffer[index-1];
                        else
                          current.carattere = buffer[index];

                        current.linea     = i;
                        break;  /* esco dal while */
                  }
      }
      else
      {
          /* carattere normale (non '\n'), incremento l'indice di posizione */
          index++;
      }

}/* fine del while */

/* ora ho in buffer la linea completa, la mia struttura e'
   inizializzata e quindi posso invertire il buffer.
   ATTENZIONE: buffer[index] = '\n'
*/
for(j=0; j<index; j++){
  buffer_inv[j] = buffer[index-j-1];
}

/* aggiungo il '\n' finale al buffer invertito */
buffer_inv[index] = '\n';

/* Ora ho tutte le strutture dati pronte, quindi mi sincronizzo
   con gli altri figli e scrivo i dati sulla pipe e sul file.
   E' importante sincronizzarsi prima di scrivere sul file per
   far si che il file finale contenga le linee in ordine giusto.
*/

/* ATTENZIONE: se sono l'ultimo figlio non ho bisogno di aspettare
   nessuno. */
if( i != (M-1) )
{
  /* leggo l'informazione dalla pipe */
  read( pipes[(i+1)%M][0],&received,sizeof(received));

  /* NOTA: siccome si e' sicuri che i non valga mai M, usare l'operatore
     modulo (%M) nell'indice dell'array delle pipe e', per questa soluzione
     irrilevante.
  */

  /* confronto l'informazione ricevuta con quella che ho calcolato */
  if( current.carattere < received.carattere ){
    current = received;
  }
}

/* scrivo su file */
printf("\nFiglio %d scrive sul file",i);
```

```
      write(fdw, &buffer_inv, index+1);

      /* scrivo sulla pipe per il figlio successivo*/
      write(pipes[i][1],&current,sizeof(current));


      /* chiudo le pipe */
      close(pipes[i][1]);
      close(pipes[(i+1)%M][0]);

      /* chiudo i file */
      close(fdr);
      close(fdw);

      /* ritorno al padre il numero di caratteri nella mia linea ('\n' escluso)*/
      exit(index);

      } /* FINE DEL CODICE DEL FIGLIO */

    }   /* fine del ciclo for di generazione dei figli */

    /* CODICE DEL PADRE */
    /* chiusura delle pipe, leggo da quella del primo figlio, ossia
       dalla pipe numero zero */
    for(i=1;i<M;i++){
      close(pipes[i][0]);
      close(pipes[i][1]);
    }

    close(pipes[0][1]);

    /* leggo l'informazione dalla pipe */
    read(pipes[0][0],&current,sizeof(current));
    printf("\nPadre ricevuta informazione %c - %d\n",current.carattere,current.linea);

    /* aspetto i figli */
    for(i=0; i<M; i++){
      pid = wait(&status);
      if( (status & 0x0FF) == 0 ){
            printf("\nFiglio PID=%d valore di ritorno %d\n",pid, ((status>>8)& 0x0FF));
      }
      else{
            printf("\nFiglio con pid %d uscito in maniera anomala",pid);
      }
    }

    return(0);
}     /* fine del main */
```