SISTEMI OPERATIVI e
LABORATORIO DI SISTEMI OPERATIVI
(A.A. 05-06) – 17 MARZO 2006

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 **ESAME17Mar06-1-1**. 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 un numero variabile di parametri che rappresentano nomi di file **F1...FN**: la lunghezza di ognuno dei file è un numero pari e multiplo intero (minore di 255) di **11**. Il processo padre deve generare **N processi figli** (**P1** … **PN**): ogni processo figlio è associato ad uno dei file **Fi**. Ognuno di tali processi figli deve creare un file il cui nome (**FCreatoi**) risulti dalla concatenazione del nome del file associato (**Fi**) con la stringa "Mezzo"; quindi, ogni figlio deve creare un **processo nipote**. La **coppia** figlio e nipote esegue concorrentemente leggendo, rispettivamente, il figlio la prima metà del file associato **Fi** e il nipote la seconda metà del file associato **Fi** (si ricordi che ogni file **Fi** ha una lunghezza pari!): la lettura deve avvenire a blocchi di 11 (si ricordi che ogni file **Fi** ha una lunghezza multiplo intero di 11!). Il processo figlio, dopo la lettura di ogni blocco **Bp** della prima metà del file, lo spedisce al processo nipote; il processo nipote, dopo la lettura di ogni blocco **Bs** della seconda metà del file, riceve il blocco **Bp** corrente dal processo figlio e scrive tale blocco **Bp** sul file **FCreatoi se e solo** se risulta **uguale** dal suo blocco corrente **Bs**. Ogni processo figlio deve ritornare al padre il numero di blocchi spediti al processo nipote.
Il padre, dopo che i figli sono terminati, deve stampare su standard output i PID di ogni figlio con il corrispondente valore ritornato.

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#define PERM 0644 /* in UNIX */

main(int argc, char **argv)
{
    /* Local variables */
    int N;                  // numero di argomenti come file
    int i;                  // variabile ausiliaria per i loop
    int pid;                // process identifier di supporto
    char *fCreato;          // il nome del file che il figlio deve creare per scriverci
    int piped[2];           // array di file descriptor per la pipe tra figlio e nipote
    int fdCreato;           // file descriptor del file creato
    int Fi;                 // file descriptor del file associato "Fi"
    char buffer[11];        // buffer in cui mettere i byte letti ad 11 alla volta
    char buffer2[11];       // buffer in cui mettere i byte letti dalla pipe
    long fileLength;        // lunghezza del file "Fi"
    int blocchi;            // contatore dei blocchi spediti dal figlio al nipote
    int status;             // variabile per lo stato della wait
    /* ----------------*/

    // Controllo sul numero dei parametri
    N=argc-1;
    if(N == 0)
    {
          printf("Errore nel numero dei parametri. Uso: %s file1 file2 ... fileN\n",argv[0]);
          exit(-1);
    }

    // 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*/
    {

          // Il figlio deve prima creare il suo file
          fCreato = (char *) malloc(strlen(argv[i+1])/*Nome file*/+5/*Mezzo*/+1/*'\0'*/);
          if(!fCreato)
          {
                printf("Errore di allocazione della memoria\n");
                exit(-3);
          }
          strcpy(fCreato, argv[i+1]);
          strcat(fCreato, "Mezzo");
          if((fdCreato=creat(fCreato, PERM))<0)
          {
                printf("Errore di creazione del file \" %s \"\n", fCreato);
```

```
	exit(-4);
}

printf("Sono figlio con PID %d e ho creato il file %s\n", getpid(), fCreato);

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

// Calcolo un'unica volta la lunghezza del file Fi associato a questa coppia
// figlio-nipote
fileLength=lseek(Fi, 0L, 2);

// Il figlio deve aprire una pipe per comunicare col nipote
if(pipe(piped)<0)
{
	printf("Errore di apertura pipe figlio/nipote\n");
	exit(-5);
}

// Il figlio di indice i deve creare un nipote
if((pid=fork())<0)
{
	printf("Errore nella creazione di un nipote\n");
	exit(-4);
}

if (pid==0) /* codice del nipote*/
{
	// Innanzitutto il nipote deve chiudere il lato di pipe che non usa
	// (scrittura)
	close(piped[1]);

	// chiudo il file aperto dal figlio
	close(Fi);

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

	printf("Sono il nipote con PID %d e comincio a leggere dal file
	%s\n",getpid(), argv[i+1]);

	// Mi sposto con l'I/O pointer sulla metà del file
	lseek(Fi, fileLength/2L , 0);

	// Devo leggere fileLength
	while(read(Fi, buffer, 11)==11)
	{
		// ho letto un blocco e devo aspettare il blocco del figlio
		read(piped[0], buffer2, 11);
```

```
                        // N.B. Usiamo il memcpm perchè la strcmp e la strncmp si fermano al
                        // terminatore, che in un buffer di memoria può presentarsi
                        if (memcmp(buffer, buffer2, 11)==0) // sono uguali i due blocchi
                              write(fdCreato, buffer, 11);
                  }

                  // fine del processo nipote
                  exit(0);
            }
            else /* codice del figlio*/
            {
                  // Innanzitutto il nipote deve chiudere il lato di pipe che non usa
                  // (lettura)
                  close(piped[0]);

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

                  // Il figlio ha già aperto il file prima di creare il nipote
                  // Quindi ora riporto l'I/O pointer all'inizio
                  lseek(Fi, 0L, 0);
                  blocchi=fileLength/22; // = (fileLength/2)/11

                  // Leggo finchè non sono alla metà del file
                  for( ; blocchi>0; blocchi--)
                  {
                        // leggo dal file Fi un blocco di 11 byte
                        read(Fi, buffer, 11);

                        // comunico al figlio i byte letti
                        write(piped[1], buffer, 11);
                  }

                  // dealloco la memoria dinamica per il nome del file creato
                  if(fCreato)
                        free(fCreato);

                  // ritorno al padre i blocchi spediti al nipote
                  exit(fileLength/22);
            }

      }

      /* codice del padre */
      for (i=0; i<N; i++)
      {
            pid=wait(&status);
            printf("Il figlio con PID %d è ritornato con valore %d\n", pid, (status>>8)&0xFF);
      }
}
```