# SISTEMI OPERATIVI e
# LABORATORIO DI SISTEMI OPERATIVI
# (A.A. 05-06) – 24 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 **ESAME24Mar06«T»-«N»**. 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 tre parametri: il primo deve essere il **nome assoluto di un direttorio** che identifica una gerarchia (**G**) all'interno del file system, il secondo parametro deve essere considerato un numero **K** maggiore di 3/4/5, mentre il terzo deve essere considerato un numero **D** strettamente positivo. Il programma deve cercare nella gerarchia **G** specificata tutti i direttori che contengono almeno **K** file la cui lunghezza in linee sia esattamente uguale a **D**. Si riporti il nome assoluto di tali direttori sullo standard output. In ogni direttorio trovato, si deve invocare la parte in C, passando come parametri i nomi dei file trovati (**F1...FM**) che soddisfano la condizione precedente e il numero **D**.

La parte in C accetta un numero variabile di parametri che rappresentano nomi di file **F1...FM** e un numero **D** strettamente positivo, che rappresenta il numero di linee di ogni file. Il processo padre deve generare **M processi figli** (**P1** … **PM**): ogni processo figlio è associato ad uno dei file **Fi**. Ognuno di tali processi figli esegue concorrentemente leggendo le linee del file ad esso associato **Fi:** dopo la lettura di ogni linea del file **Fi** ogni processo figlio deve comunicare al processo padre la lunghezza della linea corrente. Il processo padre deve riceve la lunghezza di ogni linea da tutti i figli in ordine (prima quella del primo figlio e così via fino a quella dell'ultimo figlio) e deve comunicare al figlio corrispodente se scriverla o meno su standard output: in particolare, il padre controlla se tale lunghezza è minore strettamente/maggiore strettamente/uguale/minore/maggiore di 5/6/7/8 e in questo caso comunica al figlio di scrivere la linea sullo standard output, altrimenti gli comunica di non scriverla. Ogni processo figlio deve ritornare al padre il numero di linee scritte/non scritte sullo standard output.
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 <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#define PERM 0644 /* in UNIX */

typedef int pipe_t[2];

/* ============ VARIABILI GLOBALI ============ */
char linea[256];        // buffer per la lettura delle linee
int lineeStampate;      // numero di linee stampate su stdout dal figlio
int j;                  // indice nel buffer della linea
/* ========================================== */

/* Il padre ha segnalato che bisogna scrivere la linea su stdout*/
void scrivi(int sig)
{
      write(1, linea, j);
      write(1, "\n", 1);
      lineeStampate++;

      signal(SIGUSR1, scrivi);
}

void salta(int sig)
{
      // non fa nulla
      signal(SIGUSR2, salta);
}

/*
      RIGA DI COMANDO
      argv[0] F1 F2 ... FM D
*/
main(int argc, char **argv)
{
      /* ============= Local variables =========== */
      int M;      // numero di file passati come argomenti della riga di comando
      int D;      // numero positivo passato come ultimo argomento
      int *pid;   // variabile per il process ID dei processi
      pipe_t * pipes;   // array di M pipe per comunicare con i figli
      int i;            // contatore per cicli for
      int Fi;           // file descriptor del file Fi
      int letti;        // numero dei caratteri letti dal padre
                        // dalle M pipe dai figli
      int lettura;      // variabile ausiliaria per mettere il ritorno della read() dalle pipe
      int status;       // variabile di stato per la wait
      /* ========================================= */

      if(argc < 3)
      {
            printf("Errore nel numero dei parametri. Uso: %s file1 file2..fileM D\n", argv[0]);
            exit(-1);
      }

      // Verifico che l'ultimo parametro sia un numero positivo
      D = atoi(argv[argc-1]);
      if( D<=0 || D>=255)
      {
            printf("Errore nell'ultimo parametro: \"%s\" non è un numero strettamente positivo
            o < 255 \n", argv[argc-1]);
            exit(-1);
      }

      M = argc-2; // tolgo argv[0] e D

      /* Creazione dell'array degli M pid dei figli */
      pid = (int *) malloc(M * sizeof(int));
      if(!pid)
      {
            printf("Errore nella allocazione della memoria dei pid\n");
            exit(-3);
      }
```

```c
/* Creazione delle M pipe dal padre verso i figli */
pipes = (pipe_t *) malloc(M * sizeof(pipe_t));
if(!pipes)
{
	printf("Errore nella allocazione della memoria delle pipe\n");
	exit(-3);
}

/* Creo le pipe in un ciclo for */
for(i=0; i<M; i++)
{
	if(pipe(pipes[i])<0)
	{
		printf("Errore di apertura pipe verso il figlio\n");
		exit(-4);
	}
}

/* Creo i processi figli */
for (i=0; i<M; i++)
{
	if((pid[i]=fork())<0)
	{
		printf("Errore nella creazione di un figlio\n");
		exit(-2);
	}
	else if(pid[i] == 0) /*codice del figlio*/
	{
			/* Chiudo le pipe che non utilizzo */
			for(j=0; j<M; j++)
			{
				close(pipes[j][0]);
				if(i!=j) close(pipes[j][1]);
			}

			/* Apro il file Fi in lettura */
			if((Fi=open(argv[i+1], O_RDONLY))<0)
			{
				printf("E' impossibile aprire il file %s\n", argv[i+1]);
				exit(0);
			}

			/* Installo i segnali di scrittura e di non-scrittura */
			signal(SIGUSR1, scrivi);
			signal(SIGUSR2, salta);

			j=0;
			lineeStampate=0;
			while(read(Fi,&linea[j],1)>0)
			{
				if(linea[j]=='\n')
				{
					/* comunico al padre la lunghezza della
					linea, escluso il '\n'*/
					write(pipes[i][1], &j, sizeof(int));

					/* mi metto in attesa del segnale di
					scrittura */
					pause();

					j=0;
				}
				else
					j++;
			}

			/*	Caso particolare: l'ultimo carattere del file non
				è '\n' --> Devo stampare lo stesso la linea al
				padre */
			if(j!=0)
			{
```

```
                        /* comunico al padre la lunghezza della linea
                        finale, senza aver letto il '\n' */
                        write(pipes[i][1], &j, sizeof(int));

                        /* mi metto in attesa del segnale di scrittura */
                        pause();
                  }


                  exit(lineeStampate);
            }

      } // file ciclo for

      /* Codice del padre */

      /* Chiudo i lati delle pipe che non uso */
      for(i=0; i<M; i++)
            close(pipes[i][1]);

      /* Ciclo di lettura dalle M pipe in cui scrivono i figli */
      letti=1;
      while(letti!=0)
      {
            letti=0;
            for(i=0; i<M; i++)
            {
                  if((lettura=read(pipes[i][0], &j, sizeof(int)))>0)
                  {
                        letti+=lettura;

                        if(j<5) // minore strettamente di 5
                              kill(pid[i], SIGUSR1);
                        else
                              kill(pid[i], SIGUSR2);
                  }
            }
      }

      /* I figli hanno finito di scrivere --> raccolgo i valori di ritorno*/
      i=0;
      while((pid[i]=wait(&status)) > 0)
      {
            printf("Il figlio %d ha scritto %d linee su stdout\n", pid[i], (status >>
            8) & 0xFF);
            i++;
      }

      /*  deallocazione dei buffer */
      free(pipes);
      free(pid);

}
```