SISTEMI OPERATIVI e
LABORATORIO DI SISTEMI OPERATIVI
(A.A. 17-18) – 13 GIUGNO 2018

IMPORTANTE: **LEGGERE LE INFORMAZIONI SUL RETRO DEL FOGLIO!!!**

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 un numero variabile di parametri **W+2** (con **W** maggiore o uguale a 2): il primo parametro deve essere il nome relativo semplice **D** di una directory, il secondo deve essere considerato un numero intero strettamente positivo (**Y**), mentre gli altri **W** devono essere **nomi assoluti di directory** che identificano **W** gerarchie (**G1, G2, …**) all'interno del file system. Il comportamento atteso dal programma, dopo il controllo dei parametri, è organizzato in **con W** fasi, una per ogni gerarchia.
Il programma, per ognuna delle **con W** fasi, deve esplorare la gerarchia **Gg** corrispondente - tramite un file comandi ricorsivo, **FCR.sh** - e deve contare ***per ogni gerarchia*** (**NON globalmente)** tutti i file che saranno cercati secondo quanto di seguito specificato. Il file comandi ricorsivo **FCR.sh** deve cercare tutte le directory di nome **D**: in ognuna di queste directory, si dovrà verificare che esista almeno un file **(F)** con lunghezza in linee uguale a **Y**: si deve riportare il nome assoluto di tali directory sullo standard output. Al termine della esplorazione di ogni singola gerarchia **Gg**, si deve riportare sullo standard output il numero di file (**F)** trovati in tale gerarchia e si deve invocare la parte in C, passando come parametri i nomi dei file ***trovati*** (**F1**, **F2**, ... **FN**) e il numero **Y**.

La parte in C accetta un numero variabile **N+1** di parametri (con **N** maggiore o uguale a **2**, da controllare**)** che rappresentano **N** nomi di file (**F1, F2. ... FN-1**), mentre l'ultimo rappresenta un numero intero strettamente positivo (**Y**) (da controllare) che indica la lunghezza in linee dei file: infatti, la lunghezza in linee dei file è la stessa (questo viene garantito dalla parte shell e NON deve essere controllato).
Il processo padre deve generare **N processi figli** (**P0, P1, … PN-1**): i processi figli **Pi (con i che varia da 0 a N-1)** sono associati agli **N**file **Ff** (con f= i+1). Ogni processo figlio **Pi** deve leggere tutte le **Y** linee del file associato **Ff calcolando la lunghezza di ogni linea, compreso il terminatore di linea**. I processi figli e il processo padre devono attenersi a questo **schema di comunicazione a pipeline**: il figlio **P0** comunica con il figlio **P1** che comunica con il figlio **P2** etc. fino al figlio **PN-1** che comunica con il **padre**. Questo schema a pipeline deve prevedere l'invio in avanti, **per ognuna delle Y linee dei file**, di un **array** di **strutture** dati ognuna delle quali deve contenere due campi: 1) *c1*, di tipo int, che deve contenere il pid del processo figlio; 2) *c2*, di tipo int, che deve contenere la lunghezza della linea corrente, compreso il terminatore di linea. *Gli array di strutture DEVONO essere creati da ogni figlio della dimensione minima necessaria per la comunicazione sia in ricezione che in spedizione.* Quindi la comunicazione deve avvenire in particolare in questo modo: il figlio **P0** passa in avanti (cioè comunica), per la prima linea, un array di strutture **A1**, che contiene una sola struttura con *c1* uguale al suo pid e con *c2* uguale alla lunghezza della prima linea (compreso il terminatore di linea) del file **F1**; il figlio seguente **P1**, dopo aver calcolato la lunghezza della sua prima linea (compreso il terminatore di linea) del file **F2**, deve leggere (*con una singola read*) l'array **A1** inviato da **P0** e quindi deve confezionare l'array **A2** che corrisponde all'array **A1** aggiungendo all'ultimo posto la struttura con i propri dati e la passa (*con una singola write*) al figlio seguente **P2**, etc. fino al figlio **PN-1**, che si comporta in modo analogo, ma passa al **padre**, **e così via per ognuna delle altre linee**. Quindi, al processo padre deve arrivare, **per ognuna delle Y linee dei file**, un array **AN** di **N** strutture (uno per ogni processo **P0** … **PN-1)**. Per ogni array **AN** ricevuto, il padre deve effettuare un ordinamento in senso crescente in base alle lunghezze e quindi deve riportare su standard output i dati così ordinati di ognuna delle **N** strutture insieme al **numero di linea cui si riferisce l'array e ai file cui si riferiscono i dati**.
Al termine, ogni processo figlio **Pi** deve ritornare al padre il valore intero corrisponde al proprio indice d'ordine (**i**); il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato.

IMPORTANTE:

1) Fare il login sui sistemi in modalità Linux usando il proprio **username** e **password**, aprire un browser sulla pagina ftp://lica02.lab.unimo.it/README, copiare il comando presente in un terminale ed eseguirlo rispondendo alle domande proposte: sul Desktop, viene creata automaticamente una directory **studente_1_1_XXX** al cui interno viene creato un file denominato student_data.csv che non va eliminato; infine , dopo avere copiato i propri file da chiavetta, passare in modalità testuale.
2) I file prodotti devono essere collocati nella directory **studente_1_1_XXX** dato che tale directory viene zippata e salvata automaticament sul server ad intervalli di tempo regolari. **ALLA SCADENZA DEL TEMPO A DISPOSIZIONE VERRÀ ATTIVATA UNA PROCEDURA AUTOMATICA DI ESTRAZIONE, PER OGNI STUDENTE DEL TURNO, DEI FILE CONTENUTI NELLA DIRETTORY SPECIFICATA.**
3) **NOVITÀ DALL'APPELLO DI LUGLIO 2016**: per facilitare le operazioni di stampa dei compiti sono imposte le seguenti regole per nominare i file da salvare nella directory **studente_1_1_USERNAME**:
   - FCP.sh per il file che contiene lo script principale (quello di partenza) della parte SHELL;
   - FCR.sh per il file che contiene lo script ricorsivo della parte SHELL;
   - main.c per il file che contiene il programma della parte C;
   - makefile per il file che contiene le direttive per il comando make.

   Devono essere rispettati esattamente i nomi indicati altrimenti NON si procederà alla correzione del compito!
4) NON devono essere presenti altri file con nome che termina con .sh o con .c nella directory **studente_1_1_USERNAME.**
5) Il tempo a disposizione per la prova è di **120 MINUTI** per il compito completo e di **90 MINUTI** per lo svolgimento della sola parte C.
6) Non è ammesso **nessun tipo di scambio di informazioni** né verbale né elettronico, pena la invalidazione della verifica: **all'ingresso deve essere lasciato il/i cellulare/i sulla cattedra e potranno essere ripresi solo all'uscita**.
7) L'assenza di commenti significativi verrà penalizzata, così come la mancanza del makefile!
8) **AL TERMINE DELLA PROVA È INDISPENSABILE CONSEGNARE IL TESTO DEL COMPITO (ANCHE IN CASO UNO STUDENTE SI RITIRI): IN CASO CONTRARIO, NON POTRÀ ESSERE EFFETTUATA LA CORREZIONE DEL COMPITO MANCANDO IL TESTO DI RIFERIMENTO.**
9) SI RICORDA CHE IN CASO DI ESITO INSUFFICIENTE è necessario visionare il compito prima di potersi iscrivere a qualunque appello successivo!

SE PUÒ SERVIRE RIPORTO IL SEGUENTE CODICE dai Lucidi di Fondamenti II e Lab. - Algoritmi di ordinamento (pag. 5):

```
void bubbleSort(int v[], int dim)
{ int i; bool ordinato = false;
     while (dim>1 && !ordinato)
     { ordinato = true; /* hp: è ordinato */
        for (i=0; i<dim-1; i++)
          if (v[i] > v[i+1]) /* ordinamento crescente; per
     l'ordinamento decrescente scrivere
          if (v[i] < v[i+1])  */
          {    scambia(&v[i],&v[i+1]);
               ordinato = false;
          }
        dim--;
     }
}
```