ESERCITAZIONE DI VENERDÌ 28/05/2021

1. Scrivere un programma C **9Set16.c** che risolva la parte C dell'Esame del 9 Settembre 2016: La parte in C accetta un unico parametro che rappresenta il nome assoluto di un file (**F**) (senza bisogno di controlli sul fatto che sia assoluto).
   Il processo padre deve generare **26** processi figli (**P0, P1, … P25**) tanti quanti i caratteri dell'*alfabeto inglese*: tutti i processi figli **Pi** (con i che varia da 0 a 25) sono associati all'unico file **F** e ognuno dei processi figli è associato al carattere alfabetico minuscolo corrispondente (**P0** è associato al carattere **'a'** fino a **P25** che è associato al carattere **'z'**). Ogni processo figlio **Pi** deve leggere i caratteri del file **F** cercando il carattere a lui associato **Ci** (per i=0, C0='a', … per ii=25, C25='z'). 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 **P25** che comunica con il **padre**. Questo schema a pipeline deve prevedere l'invio in avanti di un array di strutture dati ognuna delle quali deve contenere due campi: 1) *v1*, di tipo char, che deve contenere il carattere **Ci**; 2) *v2*, di tipo long int, che deve contenere il numero di occorrenze del carattere **Ci**, calcolate dal corrispondente processo. Ogni array di strutture utilizzato dai figli e dal padre deve avere dimensione fissa (26 elementi!). Quindi la comunicazione deve avvenire in particolare in questo modo: il figlio P0 passa in avanti (cioè comunica) un array di strutture A0 (di 26 elementi), che contiene una sola struttura significativa (nell'elemento di indice 0 dell'array A0) con v1 uguale a 'a' e con v2 uguale al numero di occorrenze del carattere 'a' trovate da P0 nel file F; il figlio seguente P1, dopo aver calcolato numero di occorrenze del carattere associato C1 nel file F, deve leggere (con una singola read) l'array A0 inviato da P0 e quindi deve confezionare l'array A1 che corrisponde all'array A0 aggiungendo nell'elemento di indice 1 la struttura con i propri dati e la passa (con una singola write) al figlio seguente P2, etc. fino al figlio P25, che si comporta in modo analogo, ma passa al padre. Quindi, al processo padre deve arrivare l'array A25. Il processo padre, *dopo aver ordinato tale array A25 in senso crescente rispetto al campo v2*, deve riportare i dati di ognuna delle 26 strutture su standard output insieme al **pid** e all'indice i del processo che ha generato tale struttura.
   Al termine, ogni processo figlio Pi deve ritornare al padre l'ultimo carattere letto dal file **F**; il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato sia come carattere che come valore ASCII (in decimale).
   SE PUÒ SERVIRE SI RIPORTA IL SEGUENTE CODICE dai Lucidi di Fondamenti II e Lab. - Algoritmi di ordinamento:

```
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])
     {
         scambia(&v[i],&v[i+1]);
         ordinato = false;
     }
   dim--;
}
}
```

2. **ESAME DEL 12 SETTEMBRE 2018**
   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): i primi due parametri devono essere considerati numeri interi strettamente positivi (**H** e **K**) con **H** strettamente minore di **K**, 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 **W** fasi, una per ogni gerarchia.
   Il programma, per ognuna delle **W** fasi, deve esplorare la gerarchia **Gg** corrispondente - tramite un file comandi ricorsivo, **FCR.sh** – e deve cercare tutti i direttori che contengono un **numero** di file leggibili (**F1, ... FN**) compreso fra **H** e **K**. Si riporti il nome assoluto di tali direttori sullo standard output. In ognuno di tali direttori trovati, si deve

invocare la parte in C, passando come parametri i nomi dei file trovati (**F1, ... FN**) che soddisfano la condizione precedente.

La parte in C accetta un numero variabile **N** di parametri (con **N** maggiore o uguale a **2**, da controllare**)** che rappresentano **N** nomi di file (**F1, F2. ... FN**).

Il processo padre deve generare **N processi figli Pi** (**P0** … **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). Ognuno di tali processi figli deve creare a sua volta un **processo nipote PPi (PP0 … PPN-1**) associato sempre al corrispondente file **Ff**. Ogni processo figlio **Pi** e ogni nipote **PPi** esegue concorrentemente andando a cercare nel file associato **Ff** tutte le occorrenze dei caratteri **numerici** per il figlio e tutte le occorrenze dei caratteri **alfabetici minuscoli** per il nipote. Ognuno dei processi figlio e nipote deve operare una modifica del file **Ff:** in specifico, ogni *nipote* deve trasformare ogni carattere alfabetico minuscolo nel corrispondente carattere alfabetico maiuscolo, mentre ogni *figlio* deve trasformare ogni carattere numerico nel carattere spazio. Una volta terminate le trasformazioni, sia i processi figli **Pi** che i processi nipoti **PPi** devono comunicare al padre il numero (in termini di ***long int***) di trasformazioni effettuate. Il padre ha il compito di stampare su standard output, rispettando l'ordine dei file, il numero di trasformazioni ricevute da ogni figlio **Pi** e da ogni nipote **PPi**, riportando opportuni commenti esplicativi, che devono includere anche il nome del file che è stato interessato dalle trasformazioni.

Al termine, ogni processo nipote **PPi** deve ritornare al figlio **Pi** un opportuno codice ed analogamente ogni processo figlio **Pi** deve ritornare al padre un opportuno codice; il codice che ogni nipote **PPi** e ogni figlio **Pi** deve ritornare è:

a) **0** se il numero di trasformazioni attuate è minore di 256;
b) **1** se il numero di trasformazioni attuate è maggiore o uguale a 256, ma minore di 512;
c) **2** se il numero di trasformazioni attuate è maggiore o uguale a 512, ma minore di 1024;
d) etc.

Sia ogni figlio **Pi** e sia il padre devono stampare su standard output il PID di ogni nipote/figlio e il valore ritornato.