ESERCITAZIONE 16/05/2024

NOTA BENE:

- ***a) Per ognuno dei programmi C che devono essere scritti per questa esercitazione (e per le prossime) deve essere scritto anche il makefile che produce il file eseguibile controllando tutti i possibili WARNING di compilazione (opzione –Wall) che devono sempre essere eliminati, come chiaramente anche gli errori di compilazione e di linking!***
- ***b) Una volta ottenuto l'eseguibile, va chiaramente verificato il funzionamento; in caso di passaggio di parametri va verificato anche che i controlli funzionino correttamente!***

1. Con un editor, scrivere un programma in C `9Giu14.c` che risolva la parte C dell'Esame del **9 Giugno 2014** che si riporta qui di seguito per comodità.
   La parte in C accetta un numero variabile di parametri (*maggiore o uguale a 2, da controllare*) che rappresentano **N** nomi assoluti di file **F1...FN**.
   Il processo padre deve generare **N** processi figli (**P0…PN-1**): ogni processo figlio è associato al corrispondente file. Ognuno di tali processi figli deve creare a sua volta un processo *nipote* (**PP0…PPN-1**): ogni processo nipote **PPi** esegue concorrentemente **calcolando la lunghezza in linee** del file associato usando in modo opportuno il comando-filtro *wc* di UNIX/Linux.
   Ogni processo figlio **Pi** deve convertire in termini di valore intero* (lunghezza) quanto scritto in termini di caratteri sullo standard output dal comando *wc* eseguito dal processo nipote **PPi**; quindi ogni figlio **Pi** deve comunicare tale lunghezza al padre. Il padre ha il compito di ricevere, rispettando l'ordine dei file, il valore lunghezza inviato da ogni figlio **Pi**, calcolando via via la somma (come *long int*) di tutti i valori ricevuti e stampando alla fine la somma totale (sempre come *long int*) su standard output.
   Al termine, ogni processo figlio **Pi** deve ritornare al padre il valore di ritorno del proprio processo nipote **PPi** e il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato.
   *Chiaramente il testo assume che il valore di lunghezza di ogni possibile file passato come parametro sia rappresentabile con un intero del linguaggio C; quindi la corrispondente soluzione può fare riferimento a questa ipotesi senza bisogno di specificarlo.

   OSSERVAZIONE1: Si ricorda quanto detto per la soluzione dell'esame del 5 Giugno 2015 relativamente alla pipe che serve al nipote per comunicare con il figlio: risulta più efficace che ogni figlio crei una singola pipe prima di creare il nipote, che in totale saranno N pipe, ma appunto considerate singolarmente.
   OSSERVAZIONE2: Il testo originale che trovate sul sito riporta l'indicazione del comando *wc*; infatti, all'epoca, veniva lasciato allo studente la capacità di capire che in realtà era meglio usare il comando-filtro *wc*, chiaramente inserendo un opportuno commento: in questo modo, analogamente a quanto abbiamo visto in SHELL, i nipoti scrivono semplicemente un numero sullo standard output che essendo collegato al lato di scrittura della pipe consentirà al figlio, in modo semplice, di ricevere solo questa informazione e NON anche il nome del file cui il comando *wc* si riferisce! Nei testi attuali, la specifica invece è puntuale e quindi se c'è scritto di usare un comando si dovranno passare il parametro nella EXEC, mentre se c'è scritto di usare un comando-filtro bisognerà implementare la ridirezione dello standard input!
   OSSERVAZIONE3: Fare attenzione che il nipote (cioè il comando-filtro) *wc* NON scrive un numero sullo standard input, ma un insieme di caratteri numerici che terminano con uno '\n'; quindi, ogni figlio dovrà andare a leggere tutti questi caratteri numerici, come visto nella soluzione dell'esame del 5 Giugno 2015; differentemente però dalla soluzione dell'esame del 5 Giugno 2015, tali caratteri vanno salvati in un buffer di caratteri; una volta terminata la lettura, nel buffer in corrispondenza del caratter '\n' (FARE ATTENZIONE A CALCOLARE correttamente l'indice!) si dovrà inserire il terminatore di stringa ('\0') e quindi poi la strina risultante potrà essere convertita, come usuale, con la funzione di libreria *atoi*.

2. Con un editor, scrivere un programma in C `31Mag19.c` che risolva la parte C dell'Esame del **31 Maggio 2019** che si riporta qui di seguito per comodità.
   La parte in C accetta un numero variabile **N** di parametri maggiore o uguale a 3 (*da controllare*) che rappresentano nomi assoluti di file **F1**...**FN**. Il processo padre deve generare **N** processi figli: i processi figli **Pi** sono associati agli **N** file **Fh** (con h = i+1). Ognuno di tali figli deve creare a sua volta un processo nipote **PPi**: ogni processo nipote **PPi** esegue concorrentemente e deve ordinare il file **Fh** secondo il normale ordine alfabetico, senza differenziare maiuscole e minuscole, usando in modo opportuno il comando *sort* di UNIX/Linux**.** Ogni processo figlio **Pi** deve ricevere solo la **PRIMA** linea inviata dal suo processo nipote **PPi** e deve inviare al processo padre una **struttura dati**, che deve contenere tre campi: 1) *c1*, di tipo *int*, che deve contenere il PID del nipote; 2) *c2*, di tipo *int*, che deve contenere la lunghezza della linea compreso il terminatore di linea; 3) *c3*, di tipo char[250]*, che deve contenere la linea corrente ricevuta dal nipote. Il padre deve ricevere, rispettando l'ordine dei file, le singole strutture inviate dai figli e deve stampare su standard output, per ogni struttura ricevuta, ognuno dei campi insieme al nome del file cui le informazioni si riferiscono: **si faccia attenzione al fatto che è demandato al padre il compito di trasformare, in una stringa, la linea ricevuta nel campo c3 di ogni struttura!** Al termine, ogni processo figlio **Pi** deve ritornare al padre la lunghezza della linea inviata al padre, ma non compreso il terminatore di linea e il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato.
   * **Ogni linea si può supporre che abbia una lunghezza massima di 250 caratteri, compreso il terminatore di linea e il terminatore di stringa.**

   OSSERVAZIONE: In questo caso risulta opportuno seguire il testo e quindi usare il comando *sort* (e non il comando-filtro *sort*), in modo da avere un codice del nipote con meno istruzioni, chiaramente passando il nome del file nella *execlp*!
   N.B. Sul sito si trovano altre tre versioni del testo con le rispettive soluzioni: quella riportata qui è la versione dei turni 1 e 2.

3. Con un editor, scrivere un programma in C `9Set16.c` che risolva la parte C dell'Esame del **9 Settembre 2016** che si riporta qui di seguito per comodità.
   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 **i**=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 ad '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 il 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: **← N.B. Questo codice era stato fornito insieme con il testo dell'esame!**

```
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--;
  }
}
```

OSSERVAZIONE1: Poiché, in questo caso, il numero di processi è noto **staticamente**, l'array di pipe che verrà usato in *pipeline* è da definire in modo **statico** e quindi senza bisogno di usare *malloc*; stesso discorso per l'array di strutture che deve essere usato dai figli per leggere dal precedente (a parte il primo figlio) e poi scrivere al successivo (l'ultimo scrive al padre). Ci si ricordi che, poiché il padre deve stampare i PID dei figli che hanno mandato i vari conteggi, il padre deve salvare (sempre in un array statico) i PID dei figli!
OSSERVAZIONE2: Come visto a lezione (anche se, nel caso della soluzione del 26 Maggio 2017, il numero di processi e quindi di pipe era noto *solo* dinamicamente), nel caso di comunicazione in *pipeline* ogni processo (a parte il primo) legge dalla pipe **i-1** e scrive sulla pipe **i**.
OSSERVAZIONE3: Ricordarsi che, come nel caso dei programmi `provaEsame2-a.c` e `provaEsame2-b.c` dell'esercitazione dell'11/05/2023, l'apertura dell'unico file **F** deve essere effettuata da ogni figlio e NON dal padre, dato che ogni figlio deve avere il proprio file-pointer per cercare il carattere associato!
OSSERVAZIONE4: Poiché si deve andare ad ordinare l'array ricevuto dal padre, la funzione che fa l'ordinamento (che deve essere scritta, ad esempio, prendendo ispirazione da quella riportata nel testo!) dovrà considerare che ogni singolo elemento dell'array è una *struct* con due campi e che l'ordinamento va fatto sulla base del campo *v2*.
OSSERVAZIONE5: Fare attenzione a come il padre, una volta ordinato l'array, deve calcolare l'indice dei figli e quindi il loro PID!

4. Con un editor, scrivere un programma in C `12Set18.c` che risolva la parte C dell'Esame del **12 Settembre 2018** che si riporta qui di seguito per comodità.
   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 768;
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.

OSSERVAZIONE1: In questo caso, devono essere previsti due array di pipe, uno per la comunicazione **figli-padre** e uno per la comunicazione **nipoti-padre** e queste pipe le deve *TUTTE* creare il padre prima di creare i figli!

OSSERVAZIONE2: La chiusura delle pipe nipoti-padre da parte di ogni figlio va assolutamente fatta *SOLO* dopo la creazione del proprio nipote!

OSSERVAZIONE3: Come nel programma `sostituisciCar.c` dell'esercitazione del 20 Aprile 2023, fare attenzione alla modalità di apertura del file associato, che deve essere aperto sia nel figlio che nel nipote, e di come si debba operare la trasformazione.

OSSERVAZIONE4: Fare attenzione che, nonostante ogni figlio debba aspettare il proprio nipote per stampare come richiesto PID e valore ritornato, i figli NON devono tornare al padre tale valore, ma devono tornare un proprio valore, calcolato come indicato nel testo!