ESERCITAZIONE 2/05/2024

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! Si ricorda, inoltre, di fare attenzione al fatto che se si usa un particolare standard del linguaggio, anche questo va specificato dato che non si può presupporre che chi deve verificare il funzionamento abbia lo stesso standard di default!***
- ***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!***
- ***c) Facendo riferimento anche agli esercizi della esercitazione del 25/04/2024 (data a casa) e di quelle future, e in completa analogia a quanto fatto negli esercizi SHELL, qualunque stampa (in particolare, usando la printf) che sia aggiunta rispetto a quanto richiesto dalla specifica è opportuno che venga segnalata dalla stringa "DEBUG"!***

1. Con un editor, scrivere un programma in C `prova.c` che usando una delle primitive della famiglia EXEC invochi ricorsivamente sé stesso: prima della invocazione ricorsiva, si chieda all'utente se l'invocazione **ricorsiva** vada effettuata o meno sulla base del valore intero letto dallo standard input (con una *scanf*) che se *zero* (0) significa che l'invocazione NON deve avere luogo e quindi il processo deve avere termine. Si verifichi il funzionamento fornendo alla *scanf* diversi valori prima di procedere ad inserire il valore 0 che concluderà l'esecuzione. Al termine di verifichi il valore di `$?`.
2. Con un editor, scrivere un programma in C `mylsConFork1.c` che, partendo dal programma `mylsConFork1.c` mostrato a lezione (disponibile su **GITHUB**), deve essere *invocato esattamente con 1 parametro* che deve essere il nome del file o di una directory su cui si vuole applicare il comando *ls -l*. Si verifichi il funzionamento con un numero scorretto di parametri e poi passando come parametro nomi relativi semplici, nomi relativi alla directory corrente e nomi assoluti sia di file che di directory, usando anche nomi o percorsi NON esistenti.
3. Con un editor, scrivere un programma in C `myGrepConFork-ridStError.c` che, partendo dal programma `myGrepConFork.c`[*] mostrato a lezione (disponibile su **GITHUB**), vada non solo ad implementare la **ridirezione** dello standard output ma anche quella dello **standard error** (sempre su */dev/null*), in modo che se, ad esempio, si passa il nome di un file che non esiste non si abbia una stampa a video dell'errore come invece si verificava con `myGrepConFork.c`, ma solo un riscontro dell'errore nel valore tornato al padre. Si verifichi il funzionamento con un numero scorretto di parametri e poi passando dei parametri in modo da provare i casi visti a lezione per il funzionamento di `myGrepConFork.c`.
4. Con un editor, scrivere un programma in C `myGrepConFork-ridStErrorEInput.c` che, partendo dal programma `myGrepConFork-ridStError.c` precedente, vada ad implementare la **ridirezione dello standard input** considerando il file il cui nome è passato come secondo parametro. **N.B.** In questo caso chiaramente fra i parametri della *exec* andrà passata solo la stringa da cercare e quindi il *grep* verrà utilizzato come **comando-filtro**! Si verifichi il funzionamento effettuando le stesse prove indicate precedentemente.

[*] Si ricorda che il programma `myGrepConFork.c` deve essere invocato *esattamente con 2 parametri*: il primo parametro deve essere considerata la stringa da cercare e il secondo il nome del file in cui cercare la stringa.

5. Con un editor, scrivere un programma in C `myCatConFork.c` che deve essere *invocato esattamente con 1 parametro* che deve essere il nome del file che si vuole visualizzare. Il programma, dopo il controllo sul numero di parametri, deve creare un processo figlio (come nei due esercizi precedenti). Il processo figlio, dopo aver implementato la **ridirezione dello standard input** dal file il cui nome è passato come parametro, deve invocare il programma `mycat` -mostrato a lezione (disponibile su **GITHUB** e che quindi deve essere scaricato se non già fatto!)- usando una delle primitive della famiglia EXEC. Il padre deve stampare su standard output il PID del figlio e il valore ritornato. Si verifichi il funzionamento con un numero scorretto di parametri e poi passando come parametro nomi di file esistenti e NON esistenti!
   OSSERVAZIONE: chiaramente un analogo effetto si potrebbe ottenere anche invocando direttamente il comando-filtro *cat*, ma si è preferito proporre di invocare `mycat` per differenziare questo esercizio dai due esercizi precedenti: in GITHUB sarà comunque caricata anche questa versione denominata `myCatOriginaleConFork.c`.
6. Scrivere un programma in C `padreFigliNipotiConExec.c` che deve prevedere *un numero variabile* **N** *di parametri* (*con* **N** *maggiore o uguale a 3, da controllare*) che rappresentano nomi di file (**F1**, **F2**. ... **FN**). Il processo padre deve generare **N** processi figli (**P0, P1, … PN-1**, indicati nel seguito **Pi**): i processi figli **Pi** (con **i** che varia da 0 a N-1) sono associati agli **N** file **Ff** (**con f = i+1**, quindi IN ORDINE!). Ogni processo figlio **Pi** deve, per prima cosa, *creare* un file **FOut** il cui nome deve risultare dalla concatenazione del nome del file associato **Ff** con la stringa "*.sort*". Quindi, ogni processo figlio **Pi** deve creare, a sua volta, un *processo nipote* **PPi**: ogni processo *nipote* **PPi** esegue concorrentemente e deve ordinare il file **Ff** secondo il normale ordinamento alfabetico usando in modo opportuno il **comando-filtro** *sort* di UNIX/Linux riportando il risultato di tale comando sul file **FOut**.
   Al termine, ogni processo *nipote* **PPi** deve ritornare al figlio **Pi** (che corrisponde al *proprio padre*) il valore ritornato dal comando *sort* (in caso di insuccesso nella esecuzione della primitiva *exec* deve essere tornato il valore **-1**) e, a sua volta, ogni processo figlio **Pi** lo deve ritornare al padre (*nonno* nei confronti dei *nipoti*). Il padre deve stampare, su standard output, i PID di ogni figlio con il corrispondente valore ritornato.