ESERCITAZIONE DI VENERDÌ 30/04/2021

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. Scrivere un programma in C **padreFiglioConStatus.c** che per prima cosa deve riportare su standard output il pid del processo corrente (processo padre) e poi deve creare un processo figlio: ricordarsi che il padre deve controllare il valore di ritorno della fork per assicurarsi che la creazione sia andata a buon fine. Il processo figlio deve riportare su standard output il suo pid e il pid del processo padre. Quindi, il processo figlio deve calcolare, in modo random, un numero compreso fra 0 e 99.
   Al termine, il processo figlio deve ritornare al padre il valore random calcolato e il padre deve riportare su standard output il PID del figlio e il valore ritornato.
   OSSERVAZIONE: per generare numeri random usare
   a) Chiamata alla funzione di libreria per inizializzare il seme:
   ```
   #include <time.h>
   srand(time(NULL));
   ```
   b) Funzione che calcola un numero random compreso fra 0 e n-1:
   ```
   #include <stdlib.h>
   int mia_random(int n)
   {
           int casuale;
           casuale = rand() % n;
           return casuale;
   }
   ```
2. Scrivere un programma in C **myGrepConFork-ridStError.c** che, partendo dal programma myGrepConFork.c visto a lezione, vada ad operare la ridirezione anche dello standard error.
3. Scrivere un programma in C **myGrepConFork-ridStErrorEInput.c** che, partendo dal programma myGrepConFork-ridStError.c precedente, vada ad operare la ridirezione anche dello standard input in modo che venga letto il contenuto del file il cui nome è passato come secondo parametro.
4. Scrivere un programma in C **padreFigliMultipli.c** che deve prevedere un singolo parametro che deve essere considerato un numero intero *N* che deve essere strettamente maggiore di 0 e strettamente minore di 255 e che per prima cosa deve riportare su standard output il pid del processo corrente (processo padre) insieme con il numero *N*. Il processo padre deve generare *N* processi figli. Ognuno di tali figli *Pi* deve riportare su standard output il suo pid insieme con il proprio indice d'ordine (*i*).
   Al termine, ogni processo figlio *Pi* deve ritornare al padre il proprio indice d'ordine e il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato.
   SUGGERIMENTO:

```
/* creazione figli */
for (i=0; i < N; i++)
{
    if ((pid=fork())<0)
    { <stampa e uscita> }
    else if (pid==0)
    {    /* codice figlio */
         <istruzioni specifiche del figlio>
         exit(<VALORE>);
```

OSSERVAZIONE IMPORTANTISSIMA: al di là del valore ritornato, è ASSOLUTAMENTE INDISPENSABILE che ci sia la presenza di una exit per fare in modo che il processo figlio termini e NON esegua il ciclo for, che invece deve essere eseguito solo dal padre!

}

} /* fine for */

5. Scrivere un programma in C **padreFigliConSalvataggioPID.c** che deve prevedere un singolo parametro che deve essere considerato un numero intero *N* che deve essere strettamente maggiore di 0 e strettamente minore di 155 e che per prima cosa deve riportare su standard output il pid del processo corrente (processo padre) insieme con il numero *N*. Il processo padre deve generare *N* processi figli. Ognuno di tali figli *Pi* deve riportare su standard output il suo pid insieme con il proprio indice d'ordine (*i*) e quindi deve calcolare in modo random (vedi sopra) un numero compreso fra 0 e 100+i.
   Al termine, ogni processo figlio *Pi* deve ritornare al padre il valore random calcolato e il padre deve stampare su standard output il PID di ogni figlio, insieme con il numero d'ordine derivante dalla creazione, e il valore ritornato.
6. Scrivere un programma in C **padreFigliConConteggioOccorrenze.c** che deve prevedere un numero variabile *N+1* di parametri: i primi *N* (*con N maggiore o uguale a 2, da controllare*) che rappresentano *N* nomi di file (F1, F2. ... FN), mentre l'ultimo rappresenta un singolo carattere *Cx* (*da controllare*). 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 dal file associato contando le occorrenze del carattere *Cx*.
   Al termine, ogni processo figlio Pi deve ritornare al padre il numero di occorrenze (*NOTA BENE: si può supporre minore di 255*) e il padre deve stampare su standard output il PID di ogni figlio e il valore ritornato.
7. 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): 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, 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 ordine alfabetico usando in modo opportuno il filtro *sort* di UNIX/Linux riportando il risultato di tale comando sul file *FOut*.
   Al termine, ogni processo nipote *PPi* deve ritornare al figlio il valore ritornato dal comando *sort* (in caso di insuccesso nella esecuzione del *sort* deve essere tornato il valore -1) e, a sua volta, ogni processo figlio *Pi* lo deve ritornare al padre. Il padre deve stampare, su standard output, i PID di ogni figlio con il corrispondente valore ritornato.