ESERCITAZIONE 25/04/2024

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 `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 proprio pid e il pid del processo padre. Quindi, il processo figlio deve generare, **in modo random**, un numero intero compreso fra 0 e 99. Al termine, il processo figlio deve ritornare al padre il valore random generato (sicuramente compreso fra 0 e 99) e il padre deve riportare su standard output il PID del figlio e il valore ritornato: **N.B.** *ricordarsi che il padre deve controllare sempre il valore di ritorno della wait e l'eventuale terminazione anormale del figlio!*
   OSSERVAZIONE: per generare numeri random usare quanto segue:
   a) Chiamata alla funzione di libreria srand per inizializzare il seme (da fare, ad esempio, nel padre e che richiede di aggiungere un ulteriore include rispetto ai soliti):
   ```
   #include <time.h>
   srand(time(NULL));
   ```
   b) Funzione che calcola un numero random compreso fra 0 e n-1 (da chiamare nel figlio):
   ```
   #include <stdlib.h>
   int mia_random(int n)
   {
        int casuale;
        casuale = rand() % n;
        return casuale;
   }
   ```
2. Con un editor, scrivere un programma in C `padreSenzaFigli.c` che per prima cosa deve riportare su standard output il pid del processo corrente (processo padre) e quindi deve attendere (?) la terminazione di un processo figlio, che però NON è stato creato. **N.B.** Questo esercizio serve per avere conferma della necessità di verificare sempre il valore di ritorno della *wait*!
3. Con un editor, scrivere un programma in C `padreFiglioConStatus1.c`**,** che deve essere una copia di `padreFiglioConStatus.c`. In questo nuovo programma, il padre dopo avere eseguito la prima *wait* ne vada ad eseguire un'altra per aspettare un secondo figlio, che però NON è stato creato. **N.B.** Questo esercizio serve per avere una *ulteriore* conferma della necessità di verificare sempre il valore di ritorno della *wait*!
4. Con un editor, scrivere un programma in C `provaValoriWait.c`**,** che, partendo dal programma `status1.c` mostrato a lezione (disponibile su **GITHUB**), modifichi il codice del processo figlio in modo che legga (con una *scanf*) un numero intero (`valore`); quindi il figlio deve verificare se `valore` è strettamente maggiore di 255 oppure se è negativo e in tal caso il figlio deve riportare su standard output che `valore` sarà troncato, altrimenti deve riportare che `valore` NON verrà troncato; il figlio deve tornare al padre, con la *exit*, `valore` e il padre deve stampare su standard output il pid del figlio e il valore ritornato. Si provi il funzionamento del programma fornendo via via i seguenti valori in input: **10, 125, 255, 256, 355, -1**; per quali valori avviene il troncamento?

5. Con un editor, scrivere un programma in C `padreFigliMultipli.c` che deve essere invocato *esattamente con 1 parametro* che deve essere considerato un numero intero **N** strettamente positivo e strettamente minore di **255**. Dopo aver fatto tutti i controlli necessari (sul numero dei parametri, sul loro 'tipo' e vincoli sopra indicati), per prima cosa il programma deve riportare su standard output il pid del processo corrente (processo padre) insieme con il numero **N**. Quindi, il processo padre deve generare **N** processi figli (P0, P1, … PN-1, indicati nel seguito **Pi**). Ognuno di tali figli **Pi** deve riportare su standard output il proprio pid insieme con il proprio indice d'ordine (**i**).
   Al termine, ogni processo figlio **Pi** deve ritornare al padre il proprio indice d'ordine (*sicuramente strettamente minore di 255*) e il padre deve stampare su standard output il pid di ogni figlio e il valore ritornato.
   SUGGERIMENTO PER LA CREAZIONE DEGLI N FIGLI DA PARTE DEL PADRE:

```
int N;      /* numero di figli */
int pid;    /* pid per fork */
int i;      /* indice per i figli */ ← usiamo i perché il testo parla di Pi
/* creazione figli */
for (i=0; i < N; i++)
{
        if ((pid=fork())<0)
        { <stampa e uscita> }
        if (pid==0)
        {       /* codice figlio */
                <istruzioni specifiche del figlio>
                exit(<VALORE>);
      }
} /* fine for */
```

   OSSERVAZIONE IMPORTANTISSIMA: fare attenzione che nel codice dei figli ci sia la presenza di una invocazione alla primitiva *exit* per fare in modo che ogni processo figlio termini e NON esegua il ciclo for, che invece deve essere eseguito solo dal padre!
   SUGGERIMENTO PER L'ATTESA DEGLI N FIGLI DA PARTE DEL PADRE:

```
int pidFiglio, status, ritorno;  /* per wait e valore di ritorno figli */
for (i=0; i < N; i++)
{
        pidFiglio = wait(&status);
        if (pidFiglio < 0)
        { <stampa e uscita> }
        if ((status & 0xFF) != 0)
                printf("Figlio con pid %d terminato in modo anomalo\n", pidFiglio);
        else
        {
                ritorno=(int)((status >> 8) & 0xFF);
                printf("Il figlio con pid=%d ha ritornato %d\n", pidFiglio, ritorno);
        }
}
```

N.B. Volendo dopo la *wait* si possono usare le MACRO apposite!

   ALTRA OSSERVAZIONE IMPORTANTISSIMA: fare attenzione che l'ordine con cui terminano i processi figli è completamente NON deterministico e quindi NON si potrà mai supporre con il codice mostrato precedentemente di poter riportare l'indice dei figli che via via terminano. Per avere questa informazione, considerare l'esercizio seguente!

6. Con un editor, scrivere un programma in C `padreFigliConSalvataggioPID.c` che deve essere invocato *esattamente con 1 parametro* che deve essere considerato un numero intero **N** strettamente positivo e strettamente minore di **155**. Dopo aver fatto tutti i controlli necessari (sul numero dei parametri, sul loro 'tipo' e vincoli sopra indicati), per prima cosa il programma deve riportare su standard output il pid del processo corrente (processo padre) insieme con il numero **N**. Quindi, il processo padre deve generare **N** processi figli (P0, P1, … PN-1, indicati nel seguito **Pi**). Ognuno di tali figli **Pi** deve

riportare su standard output il proprio 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.
SUGGERIMENTO PER LA CREAZIONE DEGLI N FIGLI DA PARTE DEL PADRE: Poiché il padre dove recuperare il *numero d'ordine derivante dalla creazione* è ASSOLUTAMENTE indispensabile che prima di creare i figli venga allocato un array dinamico (di dimensione **N**) in cui il padre andrà a salvare i pid dei figli e che verrà poi usato per cercare al suo interno il valore ritornato dalla wait e quindi poter stampare l'indice corrispondente che è il numero d'ordine derivante dalla creazione!

7. 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*). Dopo aver fatto tutti i controlli (quelli indicati), 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 (*supposto minore di 255*) e il padre deve stampare su standard output il pid di ogni figlio e il valore ritornato.
   SUGGERIMENTO PER CODICE FIGLI: poiché l'indice dei figli varia da 0 a N-1, quando ogni figlio dovrà identificare il file che deve essere aperto, si dovrà usare l'indice del figlio **+ 1** per selezionare il giusto nome del file da *argv*. In questo caso, possiamo delegare ai singoli figli il controllo che i singoli parametri (escludendo l'ultimo) siano nomi di file! **ATTENZIONE**: in caso di errore nella *open*, il figlio deve tornare al padre un valore che faccia capire al padre che il figlio è incorso in un errore; considerando che la specifica indica che il numero di occorrenze *può essere supposto minore di 255*, il valore tornato dai figli se la *open* fallisce può essere **-1** che verrà interpretato dal padre come **255**! Quindi ad esempio la stampa del padre può essere variata, rispetto a quanto riportato sopra, nel seguente modo:

```
printf("Il figlio con pid=%d ha ritornato %d (se 255 problemi!)\n", pidFiglio, ritorno);
```