Tree Process

Creare un'applicazione in C che gestisca un albero di processi tramite dei segnali. In particolare, il programma, una volta lanciato, deve accettare i seguenti comandi:

- **child** *n* --> crea nuovi figli al livello *n*
- **kill** n --> termina i figli al livello *n*
- **print** --> stampa in output l'albero dei processi
- **quit** --> termina il processo dopo aver terminato tutti i figli

L'intera comunicazione deve avvenire esclusivamente tramite segnali inviati dal processo principale. L'output del comando 'p' non deve essere ordinato ma deve essere ben chiaro il livello di ogni processo (per esempio usando la tabulazione).

Esempio:

```
Next command: c1
Creating child at level 1
I'm new child at level 1 with id = 1677

Next command: c1
Creating child at level 1
I'm new child at level 1 with id = 1678

Next command: c2
Creating grandchild at level 2
I'm new child at level 2 with group = 1674
I'm new child at level 2 with group = 1674

Next command: c2
Creating grandchild at level 2
I'm new child at level 2 with group = 1674
I'm new child at level 2 with group = 1674

Next command: c3
Creating grandchild at level 3
I'm new child at level 3 with group = 1675
I'm new child at level 3 with group = 1675
I'm new child at level 3 with group = 1675
I'm new child at level 3 with group = 1675

Next command: a
Invalid parameter

Next command:
```

```
Next command: p
Printing Tree:
[ID 1672 - Parent: 0] depth 0
                        [ID 1683 - Parent: 1682] depth 3
        [ID 1678 - Parent: 1672] depth 1
                        [ID 1685 - Parent: 1679] depth 3
                [ID 1680 - Parent: 1678] depth 2
                [ID 1681 - Parent: 1677] depth 2
        [ID 1677 - Parent: 1672] depth 1
                [ID 1682 - Parent: 1678] depth 2
                        [ID 1686 - Parent: 1680] depth 3
                        [ID 1684 - Parent: 1681] depth 3
                [ID 1679 - Parent: 1677] depth 2

Next command: k3

Next command: p
Printing Tree:
[ID 1672 - Parent: 0] depth 0
        [ID 1678 - Parent: 1672] depth 1
                [ID 1679 - Parent: 1677] depth 2
                [ID 1680 - Parent: 1678] depth 2
                [ID 1681 - Parent: 1677] depth 2
        [ID 1677 - Parent: 1672] depth 1
                [ID 1682 - Parent: 1678] depth 2

Next command: q
Terminating...
```

Tree process – Queue

Trasforma il programma precedente così che l'intera comunicazione avvenga tramite code di messaggi (queues).

Tree process – double – Queue

Trasformare il programma precedente per poter gestire due alberi separati. Il parametro accettato sarà dunque preceduto da 1 o 2 (esempio: **1c2** à crea figli a profondità 2 sull'albero 1)(esempio: 2p à stampa il secondo albero).

Contatore segnali

Creare un'applicazione che effettui un conteggio dei segnali SIGUSR1 e SIGUSR2 ricevuti/gestiti da processi esterni ("mittenti").

L'applicazione deve intercettare i due tipi di segnali indicati e tenere un conteggio distinto in base al processo mittente.
Il numero massimo di "mittenti" gestibili deve essere impostato con un #define.

- All'avvio il programma deve mostrare almeno il proprio PID.
- Alla ricezione di un segnale SIGUSR1 o SIGUSR2 deve mostrare un feedback con almeno il codice del segnale e il riferimento del mittente
- Alla ricezione di un segnale SIGINT o SIGTERM il programme deve terminare mostrando un report dei conteggi suddiviso per mittente (ad esempio un elenco con PID mittente, numero SIGUSR1 ricevuti, numero SIGUSR2 ricevuti)

Communication control – Pipe anonime

Creare un programma che accetti come parametro un numero di figli che andranno creati. Il processo padre deve instaurare una comunicazione bidirezionale con pipe anonime con i figli e, successivamente, accetterà in input le seguenti opzioni:

- i*n* --> sollecita il figlio *n* (dove n non è l'id quanto l'n-esimo figlio creato) ad inviargli il suo pid
- r*n* --> sollecita il figlio *n* (dove n non è l'id quanto l'n-esimo figlio creato) ad inviargli un numero random
- q --> termina il processo e tutti i figli

Il programma, quando terminato, deve aspettare la fine di tutti i figli.

Esempio:

```
michele@FISSO:~/OS/ninthLecture$ ./communicaiton.out 15
Creating 15 processes

Next command: r16
Wrong target

Next command: w3
Wrong parameter. Allowed are 'r' and 'i'

Next command: r5
Child computing random....
Child 2597 told me: '1968078301'

Next command: r15
Child computing random....
Child 2587 told me: '2146406683'

Next command: i7
Child sending own pid....
Child 2595 told me: '2595'

Next command: q
Terminated
michele@FISSO:~/OS/ninthLecture$
```