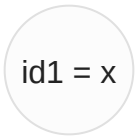
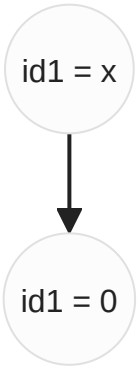
Processi multipli

Sappiamo che usando l'istruzione fork() possiamo creare un clone del processo originale, ciò ci torna utile quando vogliamo eseguire diverse parti del codice in base al processo che stiamo eseguendo.
Ma come possiamo gestire questa clonazione quando abbiamo diverse fork() che vengono chiamate?

Ipotizziamo di avere un processo e rappresentiamolo graficamente:



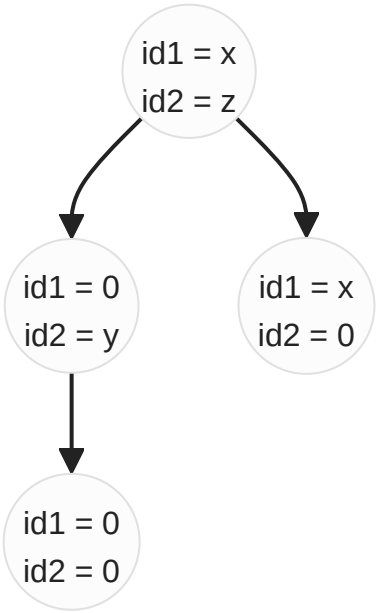
Nel momento in cui chiamiamo un fork succede questo:



Il Processo padre fa una copia del processo figlio e per la funzione fork al processo figlio verrà assegnato id uguale a 0.

> ⓘ **ID è semplicemente una variabile intera che mantiene il pid del processo, siccome i processi si evolvono in maniera diversa la variabile id1 può avere diversi valori.**

Adesso se eseguiamo nuovamente la fork(), costruiremo un processo clone per ognuno dei nuovi processi:



Ad ogni processo verrà associato un nuovo PID, i nuovi cloni erediteranno le caratteristiche dei genitori e allo stesso tempo gli verrà associato un nuovo PID uguale a 0.
Quindi se:

- Entrambi gli ID sono impostati a 0 allora sei solo un processo figlio.
- Uno dei ID è impostato allora sei un processo figlio e potresti essere anche un padre.
- Tutti gli ID sono impostati sei solo un processo padre.

Per sempliciatà chiamiamo i processi dello schema sopra (Partendo dall'alto verso il basso e da destra verso sinistra): A, B, C, D.

Di seguito un codice in C che fa capire come implementare la struttura di sopra:

```
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>


int main(int argc, char* argv[]){

        int pid1 = fork(); //2 processi
        int pid2 = fork(); //4 processi

        if(pid1 == 0){ //se vera allora siamo il figlio di un processo padre
                if(pid2 == 0){ //se vera allora siamo il figlio del processo
figlio
                        printf("Io sono il processo D");
                } else{
```

```
                        printf("Io sono il processo C");
                }
        } else{
                if(pid2 == 0){ //Siamo l'altra foglia dell'albero
                        printf("Io sono il processo B");
                } else{
                        printf("Io sono il processo A");
                }
        }
        return 0;
}
```

Adesso se volessimo fare in modo che i processi aspettino i loro processi figli prima di finire possiamo usare la funzione **wait()**, ma con attenzione, infatti questa funzione aspetta solo per un processo figlio il che crea un **problema con il processo A** che ha due figli.

> **Esempio**
>
> Se il processo C dovesse finire allora finirebbe anche il processo A, ma ciò non significa che gli altri processi siano finiti.

Ecco un modo per gestire l'uscita di più processi:

```
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>


int main(int argc, char* argv[]){

        int pid1 = fork(); //2 processi
        int pid2 = fork(); //4 processi

        if(pid1 == 0){ //se vera allora siamo il figlio di un processo padre
                if(pid2 == 0){ //se vera allora siamo il figlio del processo
figlio
                        printf("Io sono il processo D\n");
                } else{
                        printf("Io sono il processo C\n");
                }
        } else{
                if(pid2 == 0){ //Siamo l'altra foglia dell'albero
                        printf("Io sono il processo B\n");
```

```
                } else{
                        printf("Io sono il processo A\n");
                }
        }
        //Ci permette di aspettare tutti i figli
        while(wait(NULL) != -1 || errno != ECHILD){
                printf("Aspettando un figlio per finire il programma\n");
        }
        return 0;
}
```