Chapitre III : Terminaison et attente du processus

Cours Système Exploitation II

1 Linfo

ISIMM

2022/2023

- **Remarque : les variables du processus père et celles du fils**

- Une variable est toujours locale à un processus ⇒ les modifications sont toujours locales
- La variable et sa valeur sont recopiées chez l'enfant à la création
- Les variables du père et du fils sont **ensuite indépendantes**

- **Par défaut une variable n'est pas exportée**
- Une variable peut être exportée chez un enfant

- Marquer une variable comme exportée : export var
- Arrêter d'exporter une variable : unset var (détruit aussi la variable)

```
$ a="existe"
$
```

```
#! /bin/bash

b="existe"
echo "a: $a"
echo "b: $b"
a="autre chose"
```

variable.sh

Processus père : Shell

Processus fils : script Shell

```
$ a="existe"
$ ./variable.sh
a:
b: existe
$
```

Variable « a » n'est pas affichée par le processus fils car cette variable appartient au processus père (Shell)

```
#! /bin/bash

b="existe"
echo "a: $a"
echo "b: $b"
```

variable.sh

Processus père : Shell

Processus fils : script Shell

```
$ a="existe"
$ ./variable.sh
a:
b: existe
$ export a
$
```

On exporte la variable a chez le processus enfant

```
#! /bin/bash

b="existe"
echo "a: $a"
echo "b: $b"
```

variable.sh

```
$ a="existe"
$ ./variable.sh
a:
b: existe
$ export a
$ ./variable.sh
a: existe
b: existe
$
```

Le processus fils peut afficher la variable « a »

```
#! /bin/bash

b="existe"
echo "a: $a"
echo "b: $b"
```

variable.sh

Les appels système wait(), waitpid() et exit()

Le processus père et le processus fils s'exécutent d'une façon parallèle.

Impossible de prévoir l'ordre du déroulement. 

→ Il faut synchroniser le déroulement des processus père et fils

Endormir un processus et lancer l'autre.

La primitive sleep() peut assurer une synchronisation d'exécution entre les processus et ordonner le déroulement du programme.

Les appels système wait(), waitpid() et exit()

Exemple1.c : Utilisation de sleep()

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
    int pid;
    printf("Je suis %d le père, je vais enfanter.\n", getpid());
    pid = fork();
    if(pid==0)
    {
        printf("Bonjour. Je suis %d le fils\n",getpid());
       sleep(2);/*Bloque le processus pendant 2 s (endormir)*/
        printf("Quel beau monde !!\n");
       sleep(4);/*Bloque le processus pendant 4 s (endormir)*/
        printf("Je ne vais plus mourir\n");

    }
    else
    {
        sleep(5);/*Bloque le processus pendant 5 s (endormir)*/
        printf("Je vais mourir moi %d\n", getpid());
       sleep(3);/*Bloque le processus pendant 3 s (endormir)*/
        printf("Mon heure est venue: Adieu %d mon fils\n",pid);
    }
}
```

Résultats

Bonjour, Je suis 2019 le père, je v

enfanter.

//Attente de 5s du père

Bonjour, Je suis 2020 le fils.

//Attente de 2s du fils

Quel beau monde !!

//Attente de 4s du fils

Je vais mourir moi 2019

//Attente de 3s du père

Je ne vais plus mourir.

Mon heure est venue. Adieu 2020 mon fil

Les appels système wait(), waitpid() et exit()

Il faut synchroniser les fins d'exécution des processus afin d'éviter les processus zombies.

→ En communiquant certaines données:

- La notification de la fin du processus fils (code du retour du processus fils)

- L'obligation de l'attente du processus père

Les appels système wait(), waitpid() et exit()

Un processus se termine :

- Par une demande d’arrêt volontaire (appel système exit() ou retrun)

- Par un arrêt forcé provoqué par un autre processus (appel système kill())

Les appels système wait(), waitpid() et exit()

Exemple2.c (primitive kill())

```
include <unistd.h> //Pour l'instruction fork();
include <stdio.h>
include <signal.h> //pour l'instruction kill et le signal SIGKILL
nt main()

   int i=0;
   int pid;
   char c;

   pid=fork();
   if (pid==0)
   {
         printf("Je suis le fils Num %d\n", getpid());
         printf("Mon pere %d va me tuer\n",getppid());
         while(1)
             printf("Je suis encore vivant\n");
   }
   else
   {
   sleep(1);
   printf("Je suis le pere Num %d\n",getpid());
   printf("Je vais tuer mon fils %d\n",pid);
   kill(pid,SIGKILL);
   printf("Mon fils est mort, je vais mourir moi %d\n",getpid());
   }
```

Résultats

```
//Endormir le père 1s
Je suis le fils Num 8173
Mon pere 8172 va me tuer
Je suis encore vivant
Je suis encore vivant
Je suis le pere Num 8172
Je suis encore vivant
Je suis encore vivant
Je vais tuer mon fils 8173
//le processus père tue le processus fils

Mon fils est mort, je vais mourir moi 8172
```

Les appels système wait(), waitpid() et exit()

#include <stdlib.h>
Void exit (int status);

La primitive **exit(status)** permet d'arrêter explicitement l'exécution d'un processus.

Le paramètre **status** spécifie un code de retour, entre 0 et 255, à renvoyer au processus père.

Par convention, en cas de **terminaison normale**, un processus doit retourner la **valeur 0**.

Avant de terminer l'exécution du processus, exit() exécute les fonctions de « nettoyage » des librairies standards : toutes les ressources systèmes qui lui ont été allouées sont libérées.

Les appels système wait(), waitpid() et exit()

Lorsqu'un processus fils se termine (exit()), le système supprime tout son contexte, sauf son entrée de la table des processus. Il devient zombie.

Le processus père, par un appel de la primitive wait(), « récupère » la mort de son fils, supprime l'entrée de la table des processus concernant son fils achevé.

→ Le processus fils à l'état zombie disparaît complétement.

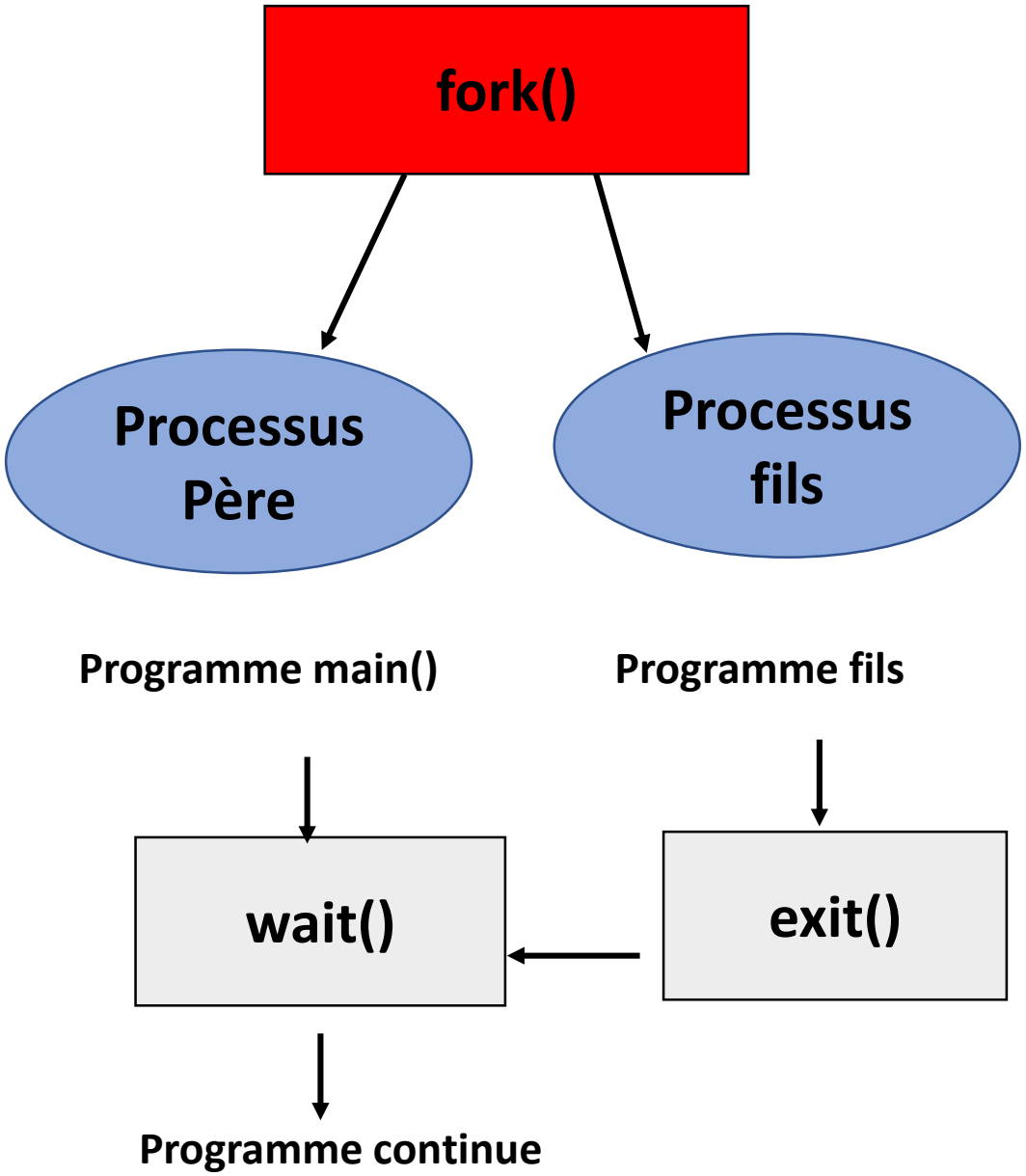
→ Il faut prendre garde de l'appel wait qui est bloquant.
→ L'exécution du père est suspendue jusqu'à ce qu'un fils se termine
→ Il faut mettre autant d'appels de wait qu'il y a de fils

Les appels système wait(), waitpid() et exit()

Le processus père, par un appel de la primitive wait(), « récupère » la mort de son fils, supprime l'entrée de la table des processus concernant son fils achevé.

→ Le processus fils à l'état zombie disparaît complétement.

→ Il faut prendre garde de l'appel wait qui et bloquant.
→ L'exécution du père est suspendue jusqu'à ce qu'un fils se termine
→ Il faut mettre autant d'appels de wait qu'il y a de fils



Les appels système wait(), waitpid() et exit()

Attente d'un processus fils : wait()

```
#include <sys/types.h>
#include <sys/wait.h>

Pid_t wait(int *status);
```

La primitive wait() suspend l'exécution du processus appelant jusqu'à ce que l'un de ses fils termine. Wait() renvoie :

- Le PID du processus fils terminé.
- La valeur -1, en cas d'erreur (dans le cas où le processus n'a pas de fils).
- Le paramètre status correspond au code de retour du processus fils qui va se terminer.
- Ce code est généralement indiqué avec la fonction exit().

Les appels système wait(), waitpid() et exit()

Exemple3.c : Un programme C qui permet à un processus père **d'attendre la terminaison de son fils**.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h> //pour wait
#include <errno.h> /* permet de récupérer les codes d'erreur */
int main(){
    pid_t pid;
    int status;
    switch (pid=fork()){
        case -1 : perror("Problème dans fork()\n");
            exit(errno); /* retour du code d'erreur */
            break;
        Case 0 :
            Sleep(2) ;
            puts("Je suis le fils");
            exit(0);
        default : puts("Je suis le père");
            puts("J'attends la fin de mon fils.");
            wait();
            printf("Mon fils %d est terminé.\n", pid);
            break;
    }
    return 0;
}
```

Résultats

```
//Endormir le fils 2s
Je suis le père
J'attends la fin de mon fils.
//Attendre l'exécution du fils
Je suis le fils
//Fin du fils
Mon fils 11100 est terminé.
```

Les appels système wait(), waitpid() et exit()

Exemple4.c : Un programme C qui permet à un processus père de **récupérer le PID et le code de retour** renvoyé par son fils dans la fonction exit.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
  pid_t pid = fork();
  if(pid == 0) {
    printf("Je suis le fils. PID=%d.\n", getpid());
    sleep(1);
    printf("Je retourne la valeur 4.\n") ;
    return 4;
  } else if (pid>0) {
    printf("Je suis le pere. PID=%d.\n", getpid());
    int status;
    pid_t pid_retour = wait(&status); //récupérer le PID du fils et son code de retour
    printf("Mon fils %d se termine avec le code de retour %d\n", pid_retour, WEXITSTATUS(status));

  } else {
    printf("probleme de creation de Fork\n");
  }
  return EXIT_SUCCESS;
}
```

On récupère la valeur 4 dans status

Les appels système wait(), waitpid() et exit()

Le père par un wait(), récupère la mort de son fils.

- Dans le cas de plusieurs fils, comment on procède si le père veut attendre la fin d'un processus fils particulier ?

#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid (pid_t pid, int *status, int options);

Les appels système wait(), waitpid() et exit()

La primitive **waitpid()** permet au processus père d'attendre un fils particulier

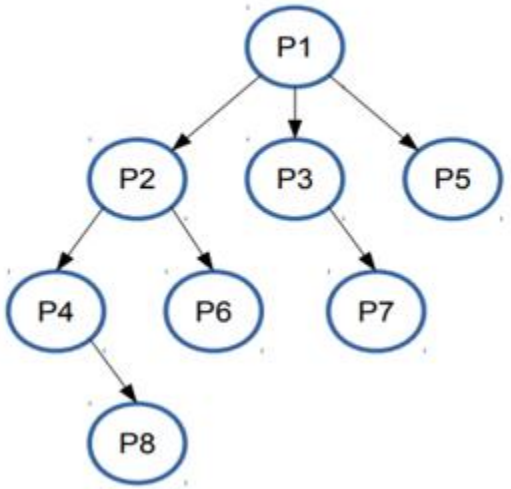
- **waitpid()** renvoie :

- Le PID du fils en cas de réussite

- -1 en cas d'échec

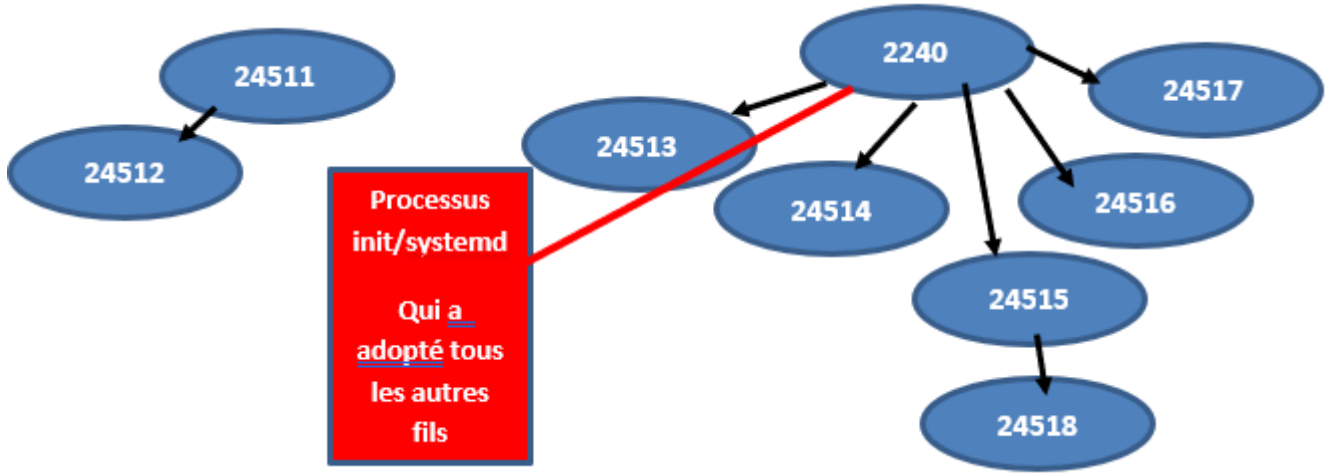
- **Cas structure en arbre**

```c
#include <stdio.h> //printf
#include <stdlib.h>//exit
#include <unistd.h> //getpid, fork
#include <sys/wait.h> //wait

int main() {
    //pid_t pid; // identifiant du processus
    int i, n;

    printf("lancer la structure en arbre, donner le nombre n \n");
    scanf("%d", &n);
    //pid = fork();
    printf("je suis le processus pere avec pid %d \n", getpid());

    for (i =0; i<n;i++)
    {
      if (fork() == 0) {

        printf("je suis le processus fils numero %d avec pid %d et mon père est %d\n", i, getpid(),getppid());
      }
      else {wait(NULL);}
    }


    return 0;
    }
```



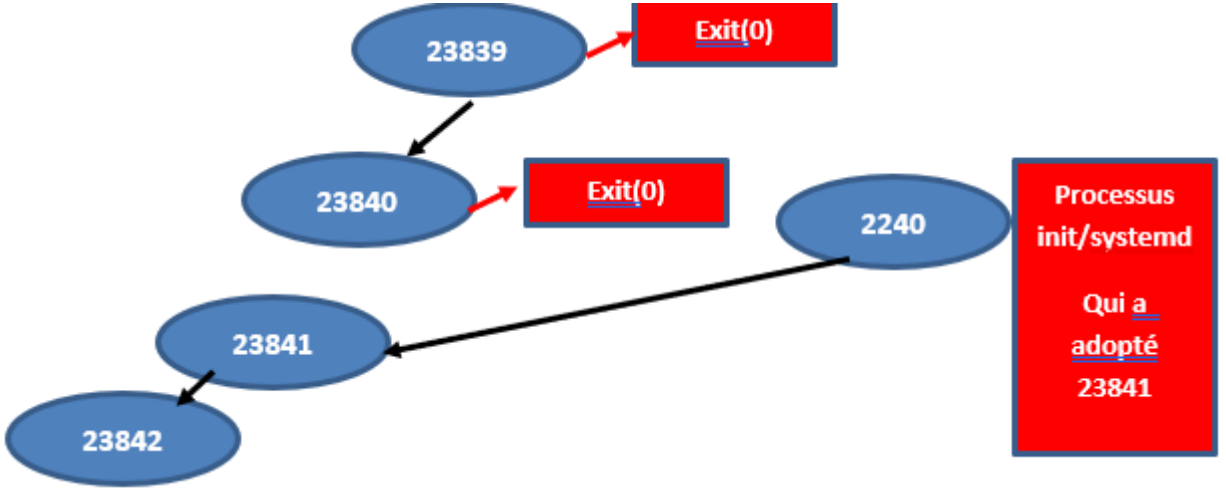

```
lancer la structure en arbre, donner le nombre n
3
je suis le processus pere avec pid 47999
je suis le processus fils numero 0 avec pid 48000 et mon père est 47999
je suis le processus fils numero 1 avec pid 48001 et mon père est 48000
je suis le processus fils numero 2 avec pid 48002 et mon père est 48001
je suis le processus fils numero 2 avec pid 48003 et mon père est 48000
je suis le processus fils numero 1 avec pid 48004 et mon père est 47999
je suis le processus fils numero 2 avec pid 48005 et mon père est 48004
je suis le processus fils numero 2 avec pid 48006 et mon père est 47999
sana@ubuntu:~/Desktop/tpFork$
```

```c
#include <stdio.h> //printf
#include <stdlib.h>//exit
#include <unistd.h> //getpid, fork
#include <sys/wait.h> //wait

int main() {
    //pid_t pid; // identifiant du processus
    int i, n;

    printf("lancer la structure en arbre, donner le nombre n \n");
    scanf("%d", &n);
    //pid = fork();
    printf("je suis le processus pere avec pid %d \n", getpid());

    for (i =0; i<n;i++)
    {
      if (fork() == 0) {

        printf("je suis le processus fils numero %d avec pid %d et mon père est %d\n", i, getpid(),getppid());

      }
```




```
lancer la structure en arbre, donner le nombre n
3
je suis le processus pere avec pid 24511
je suis le processus fils numero 0 avec pid 24512 et mon père est 24511
sana@ubuntu:~/Desktop/tpFork$ je suis le processus fils numero 2 avec pid 24514
et mon père est 2240
je suis le processus fils numero 1 avec pid 24513 et mon père est 2240
je suis le processus fils numero 2 avec pid 24516 et mon père est 2240
je suis le processus fils numero 2 avec pid 24517 et mon père est 2240
je suis le processus fils numero 1 avec pid 24515 et mon père est 2240
je suis le processus fils numero 2 avec pid 24518 et mon père est 24515
```

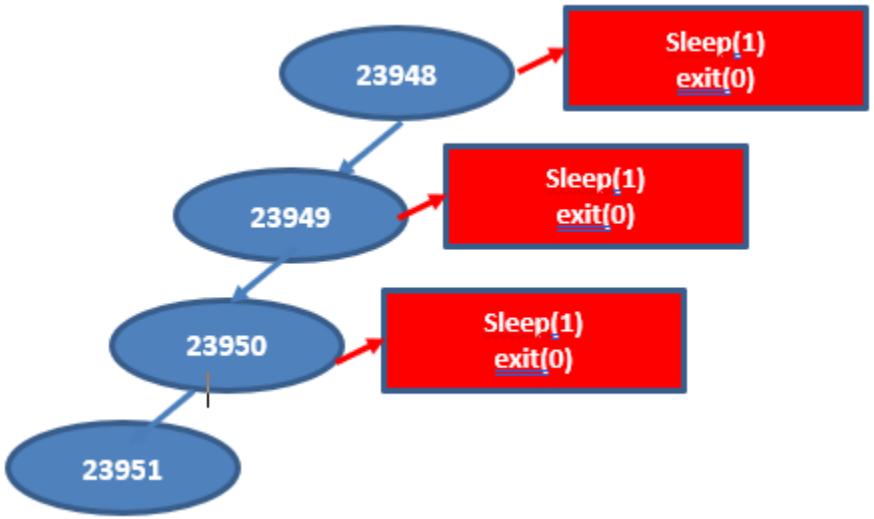
- **Sans wait(NULL) ou sleep () chez le père**
- **Avec exit(0) chez le père**



- **Par exemple pou n=3, Ici le père sera mort (retour vers le terminal) et il y a un fils qui est adopté par le processus 2240 (processus init = systemd)**

```
lancer la structure en arbre, donner le nombre n
3
je suis le processus pere avec pid 23839
je suis le processus fils numero 0 avec pid 23840 et mon père est 23839
sana@ubuntu:~/Desktop/tpFork$ je suis le processus fils numero 1 avec pid 23841
et mon père est 2240
je suis le processus fils numero 2 avec pid 23842 et mon père est 23841
```

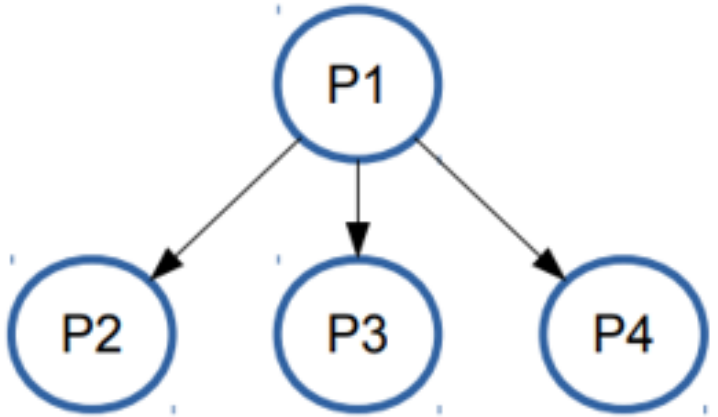
```c
#include <stdio.h> //printf
#include <stdlib.h>//exit
#include <unistd.h> //getpid, fork
#include <sys/wait.h> //wait

int main() {
    //pid_t pid; // identifiant du processus
    int i, n;

    printf("lancer la structure en arbre, donner le nombre n \n");
    scanf("%d", &n);
    //pid = fork();
    printf("je suis le processus pere avec pid %d \n", getpid());

    for (i =0; i<n;i++)
    {
      if (fork() == 0) {

        printf("je suis le processus fils numero %d avec pid %d et mon père est %d\n", i, getpid(),getppid());

      }

      else {//wait(NULL);  // le père va attendre l'un des fils
            sleep(1);     // endormir chaque processus père pendant 1s

            exit(0); } //sortie de chaque processus père
    }
    //}
```




```
lancer la structure en arbre, donner le nombre n
3
je suis le processus pere avec pid 23948
je suis le processus fils numero 0 avec pid 23949 et mon père est 23948
je suis le processus fils numero 1 avec pid 23950 et mon père est 23949
je suis le processus fils numero 2 avec pid 23951 et mon père est 23950
sana@ubuntu:~/Desktop/tpFork$
```

- **Cas structure en chaine**

```c
#include <stdio.h> //printf
#include <stdlib.h>//exit
#include <unistd.h> //getpid, fork
#include <sys/wait.h> //wait

int main() {
    //pid_t pid; // identifiant du processus
    int i, n;

    printf("lancer la structure en chaine, donner le nombre n \n");
    scanf("%d", &n);

    printf("je suis le processus pere avec pid %d \n", getpid());

    for (i =0; i<n;i++)
    {
      if (fork() == 0) {

        printf("je suis le processus fils numero %d avec pid %d et mon père est %d\n", i, getpid(),getppid());
        exit(0);
      }
      else {wait(NULL);}
    }


    return 0;
    }
```




```
lancer la structure en chaine, donner le nombre n
3
je suis le processus pere avec pid 48176
je suis le processus fils numero 0 avec pid 48177 et mon père est 48176
je suis le processus fils numero 1 avec pid 48178 et mon père est 48176
je suis le processus fils numero 2 avec pid 48179 et mon père est 48176
```

```c
#include <stdio.h> //printf
#include <stdlib.h>//exit
#include <unistd.h> //getpid, fork
#include <sys/wait.h> //wait

int main() {
    //pid_t pid; // identifiant du processus
    int i, n;

    printf("lancer la structure en chaine, donner le nombre n \n");
    scanf("%d", &n);

    printf("je suis le processus pere avec pid %d \n", getpid());

    for (i =0; i<n;i++)
    {
      if (fork() == 0) {

        printf("je suis le processus fils numero %d avec pid %d et mon père est %d\n", i, getpid(),getppid());
        //exit(0);
      }
      else {wait(NULL);}
    }


    return 0;
    }
```

```
lancer la structure en chaine, donner le nombre n
3
je suis le processus pere avec pid 24660
je suis le processus fils numero 0 avec pid 24661 et mon père est 24660
je suis le processus fils numero 1 avec pid 24662 et mon père est 24661
je suis le processus fils numero 2 avec pid 24663 et mon père est 24662
je suis le processus fils numero 2 avec pid 24664 et mon père est 24661
je suis le processus fils numero 1 avec pid 24665 et mon père est 24660
je suis le processus fils numero 2 avec pid 24666 et mon père est 24665
je suis le processus fils numero 2 avec pid 24667 et mon père est 24660
sana@ubuntu:~/Desktop/tpFork$
```

- **Si on enlève exit(0) chez les fils, on revient à la structure arbre complète**

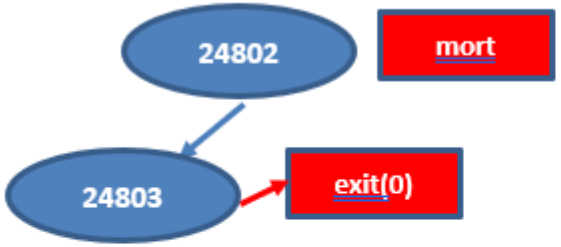
```c
#include <stdio.h> //printf
#include <stdlib.h>//exit
#include <unistd.h> //getpid, fork
#include <sys/wait.h> //wait

int main() {
    //pid_t pid; // identifiant du processus
    int i, n;

    printf("lancer la structure en chaine, donner le nombre n \n");
    scanf("%d", &n);

    printf("je suis le processus pere avec pid %d \n", getpid());

    for (i =0; i<n;i++)
    {
      if (fork() == 0) {

        printf("je suis le processus fils numero %d avec pid %d et mon père est %d\n", i, getpid(),getppid());
        //exit(0);
      }
      else {//wait(NULL);}
    }
}

    return 0;
```

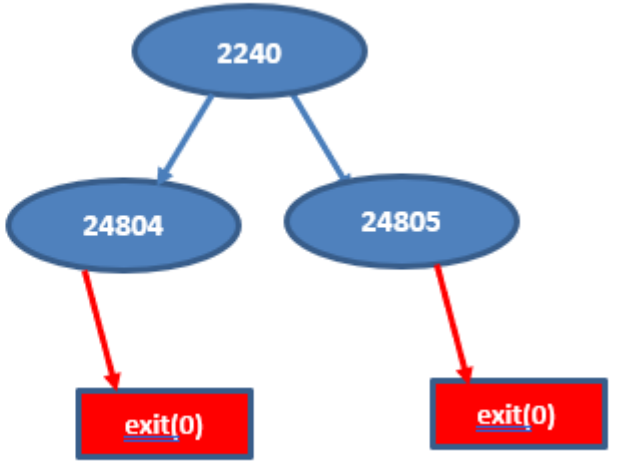
```
lancer la structure en chaine, donner le nombre n
3
je suis le processus pere avec pid 24733
je suis le processus fils numero 0 avec pid 24734 et mon père est 24733
sana@ubuntu:~/Desktop/tpFork$ je suis le processus fils numero 2 avec pid 24736
et mon père est 2240
je suis le processus fils numero 1 avec pid 24737 et mon père est 2240
je suis le processus fils numero 2 avec pid 24738 et mon père est 2240
je suis le processus fils numero 1 avec pid 24735 et mon père est 2240
je suis le processus fils numero 2 avec pid 24739 et mon père est 2240
je suis le processus fils numero 2 avec pid 24740 et mon père est 24737
```

- **Si on enlève exit(0) chez les fils et wait(NULL) chez le père, on aura une structure incompréhensible car le père sera mort et quelques fils seront adoptés par le processus init**

```c
#include <stdio.h> //printf
#include <stdlib.h>//exit
#include <unistd.h> //getpid, fork
#include <sys/wait.h> //wait

int main() {
    //pid_t pid; // identifiant du processus
    int i, n;

    printf("lancer la structure en chaine, donner le nombre n \n");
    scanf("%d", &n);

    printf("je suis le processus pere avec pid %d \n", getpid());

    for (i =0; i<n;i++)
    {
      if (fork() == 0) {

        printf("je suis le processus fils numero %d avec pid %d et mon père est %d\n", i, getpid(),getppid());
        exit(0);
      }
      else {//wait(NULL);}
    }
}

    return 0;
    }
```

```
lancer la structure en chaine, donner le nombre n
3
je suis le processus pere avec pid 24802
je suis le processus fils numero 0 avec pid 24803 et mon père est 24802
je suis le processus fils numero 1 avec pid 24804 et mon père est 2240
sana@ubuntu:~/Desktop/tpFork$ je suis le processus fils numero 2 avec pid 24805
et mon père est 2240
```





- **Si on laisse exit(0) chez les fils et on enlève wait(NULL) chez le père**
- **Le père est mort sans attendre ses fils (il créé un seul fils et meurt), les autres fils sont adopté par le processus 2240**