

Manipulation des appels système fork, pipe

■ ■ ■ Manipulation de fork & pipe

1 – Il y a 8 affichage de « Salut ».

2 – Combien de lignes « salut » affiche chacun de ces programmes :

Programme 1 :

```
1 int main() {
2     int i;
3     for (i=0; i<2; i++)
4         fork();
5     printf("salut\n");
6     exit(0);
7 }
```

Il y a 4 affichages.

Programme 2 :

```
1 void go() {
2     fork();
3     fork();
4     printf("salut\n");
5 }
6 int main() {
7     go();
8     printf("salut\n");
9     exit(0);
10 }
```

Il y a 8 affichages.

Programme 3 :

```
1 int main() {
2     if (fork())
3         fork();
4     printf("salut\n");
5     exit(0);
6 }
```

Il y a 3 affichages.

Programme 4 :

```
1 int main() {
2     if (fork() == 0)
3     { if (fork())
4         { printf("salut\n");
5         }
6     }
7 }
8 }
```

Il y a 1 affichage.

3 – Écrire un programme qui crée 10 processus fils :

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

int main()
{
    for (int i = 0; i <10; i++)
    {
        if (fork()==0)
        {
            // je suis l'enfant
            for(int j = 0; j < 10; j++)
            {
                printf("%d\n", i);
            }
            exit(0);
        }
    }
    for (int i = 0; i <10; i++)
        wait(NULL);
}
```

#### 4 – Crible d’Erathostène :

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

#define LECTURE 0
#define ECRITURE 1

int main()
{
    int i;
    int tube_succeesseur[2];

    pipe(tube_succeesseur);
    if (fork() != 0)
    {
        /* je suis le pere */
        close(tube_succeesseur[LECTURE]);
        for(i = 2; i < 100; i++)
            write(tube_succeesseur[ECRITURE], &i, sizeof(int));
        close(tube_succeesseur[ECRITURE]);
        wait(NULL); /* On attend la fin du premier fils */
    }
    else
    {
        /* je suis le fils */
        int premier, tube_predecesseur[2], valeur, existe_succeesseur = 0;

        close(tube_succeesseur[ECRITURE]);
        tube_predecesseur[LECTURE] = tube_succeesseur[LECTURE];
        read(tube_predecesseur[LECTURE], &premier, sizeof(int));
        printf("Valeur premiere : %d\n", premier);

        while (1) /* Boucle infinie, sortie est prévue lors de la fermeture du tube */
        {
            int resultat;

            resultat = read(tube_predecesseur[LECTURE], &valeur, sizeof(int));
            if (!resultat)
                exit(0); /* Sortie lorsque le tube est fermé, et la lecture impossible */
            if ((valeur % premier) != 0)
            {
                if (existe_succeesseur)
                {
                    write(tube_succeesseur[ECRITURE], &valeur, sizeof(int));
                }
                else
                {
                    pipe(tube_succeesseur);
                    if (fork() != 0)
                    {
                        /* je suis le pere */
                        close(tube_succeesseur[LECTURE]);
                        existe_succeesseur = 1;
                        write(tube_succeesseur[ECRITURE], &valeur, sizeof(int));
                    }
                    else
                    {
                        /* je suis le fils */
                        close(tube_predecesseur[LECTURE]);
                        close(tube_succeesseur[ECRITURE]);
                        tube_predecesseur[LECTURE] = tube_succeesseur[LECTURE];
                        read(tube_predecesseur[LECTURE], &premier, sizeof(int));
                        printf("Valeur premiere : %d\n", premier);
                    }
                }
            }
        }
    }
}
```

## ■ ■ ■ Manipulation des signaux

### 5 – Écrire un programme P créant deux fils :

- ▷ P envoie le signal SIGUSR1 à son second fils ;
- ▷ à la réception de ce signal, le second fils envoie le signal SIGUSR2 au premier fils (qui provoque sa terminaison) avant de s'arrêter.

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

int tableau_pid_enfants[2];
int compteur = 0;

void gestion_signal_enfant0(int n)
{
    printf("Enfant 1, %d, reception SIGUSR2\n", getpid());
    exit(0);
}

void gestion_signal_enfant1(int n)
{
    printf("Enfant 2, reception SIGUSR1\n");
    printf("Envoi SIGUSR2 vers %d\n", tableau_pid_enfants[0]);
    kill(tableau_pid_enfants[0], SIGUSR2);
    exit(0);
}

void attente_parent(int n)
{
    printf("+");
    compteur++;
}

int main()
{
    signal(SIGUSR1, attente_parent);
    for(int i=0; i<2; i++)
    {
        int pid = fork();
        if (pid != 0)
        { // je suis le parent
            printf("Parent lance %d\n", pid);
            tableau_pid_enfants[i] = pid;
        }
        else
        { // je suis l'enfant
            char c = '\0';
            if (i == 0) signal(SIGUSR2, gestion_signal_enfant0);
            if (i == 1) signal(SIGUSR1, gestion_signal_enfant1);

            kill(getppid(), SIGUSR1);
            scanf("%c", &c);
            printf("%d : Je ne devrais pas mais je finis la\n", getpid());
            exit(0);
        }
    }
    while(compteur!=2);
    printf("Parent envoi signal SIGUSR1 a %d\n", tableau_pid_enfants[1]);
    kill(tableau_pid_enfants[1], SIGUSR1);
    for(int i=0; i<2; i++)
        wait(NULL);
    printf("Parent fin\n");
}
```

6 – Écrire un programme qui se termine uniquement au 5<sup>ème</sup> «Ctrl-C».

Avec l'utilisation de «sigaction» :

En comptant 5 :

```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 void countFive (int signal) {
7     static int i=1;
8
9     if (i == 5)
10        exit (0);
11    i++;
12 }
13
14 int main () {
15     struct sigaction nvt, old;
16     memset (&nvt, 0, sizeof(nvt));
17     nvt.sa_handler=countFive;
18     sigaction (SIGINT, &nvt, &old);
19     for (;;)
20 }
```

En comptant 4 et en désactivant le support du signal :

```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 struct sigaction nvt, old;
7
8 void countFive (int signal) {
9     static int i=1;
10
11    if (i == 4)
12        sigaction (SIGINT, &old, NULL);
13    i++;
14 }
15
16 int main () {
17     memset (&nvt, 0, sizeof(nvt));
18     nvt.sa_handler=countFive;
19     sigaction (SIGINT, &nvt, &old);
20     for (;;)
21 }
```

Avec l'utilisation de «signal» :

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

void gestion_sigint(int n)
{
    static int compteur = 0;

    compteur++;
    printf("\ncompteur=%d\n", compteur);

    /* Utile pour savoir où se trouve la variable compteur */
    /* printf("adresse: %p\n",&compteur); */

    if (compteur == 5)
        exit(0);

    printf("Meme pas mal\n");
}

int main()
{
    char c;
    signal(SIGINT, gestion_sigint);
    printf("starting");
    scanf("%c", &c);
}
```