TP № 2: Processus

Samir Aknine, Antoine Gréa & **Rémy Grünblatt** 08/04/2020

Compétences à acquérir:

- Maîtriser le cycle de vie des processus
- Savoir créer des processus en C
- Tuer des processus zombies

Information: Il est extrement conseillé d'avoir recourir à une machine sous GNU/Linux (par exemple, Ubuntu). Il est possible d'utiliser https://shell.univ-lyon1.fr en secours, en ouvrant plusieurs terminaux (notamment dans le cas du processus zombie), mais cela ne devrait pas être la première option pour vous.

Important: Tous les TPs feront l'objet de rendus, et seront notés; les modalités de rendu seront précisées par email ou sur le chat discord de la classe. Il convient donc de bien noter les commandes exécutées, par exemple dans un fichier texte ou markdown, et leur résultats, afin de préparer le rendu du TP.

Exercice 1: Fork Simple

On donne le code source C suivant:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    int pid;
    printf("debut\n");
    pid = fork();
    if (pid == 0) {
        printf("execution 1\n");
    } else {
        printf("execution 2\n");
    }
    printf("Fin\n");
    return 0;
}
```

Information: Il est possible d'utiliser la commande « `man 2 fork` » pour obtenir la page de manuel liée à fork, dans la section 2. Les sections du manuel (documentées dans « `man man` ») sont les suivantes:

1. User Commands
2. System Calls
3. C Library Functions
4. Devices and Special Files

...

Ainsi, fork étant un appel système, il faudra écrire « `man 2 fork` », mais si l'on cherche la documentation de printf (la fonction C), il faudra utiliser « `man 3 printf` ».

Question 1: Théorie

Prévoir le résultat de l'exécution de ce programme, et donner d'arbre des processus de ce programme.

Question 2: Pratique

Compiler le programme en terminal, à l'aide de la commande « `gcc mon_programme.c -o nom_du_binaire` ». Exécuter le programme et vérifier la réponse précédente, en modifiant par exemple le programme pour qu'il affiche le PID du processus appelant.

Exercice 2: Fork Imbriqués

On donne le code source C suivant:

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

int main(int argc, char *argv[]) {
    int a, e;
    a = 10;
    if (fork() == 0) {
        a = a * 2;
        if (fork() == 0) {
            a = a + 1;
            exit(2);
```

```
        }
        printf("%d \n", a);
        exit(1);
    }
    wait(&e);
    printf("a : %d ; e : %d \n", a, WEXITSTATUS(e));
    return(0);
}
```

Question 1: Arbre des processus

- Donnez l'arbre des processus de ce programme. Indiquez ce que chaque processus affiche.
- On supprime désormais l'instruction « `exit(2)` ». Reprendre la question précédente avec cette modification.

Question 2: Processus Zombie

Téléchargez le programme « zombie » disponible à l'adresse https://remy.grunblatt.org/vrac/. Le programme zombie grogne quand il reçoit un signal autre que SIGBUS. Il est également responsable de l'affichage que vous pouvez constater quand vous l'exécutez dans un terminal. Attention ne fermez pas le terminal au risque de rendre l'exercice bien plus compliqué !

- Créer un programme nommé « `killbill` » (écrit en C) qui est capable de tuer le processus zombie ainsi créé. La fonction disponible dans le fichier « parentpid.h » vous sera très probablement utile.
- Pourquoi le zombie grogne-t-il quand il se prend le bus ? Précisez la nature de ce phénomène.

Bonus facultatif: écrire « `killbill` » en bash / shell.

Exercice 3: La primitive « `exec` »

On donne le code source C suivant:

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

int main() {
```

```
    int p;
    p = fork();
    if (p == 0) {
        sleep(2);
        execl("/bin/echo", "echo", "Nonnnnnn", "!", NULL);
    }
    wait(NULL);
    printf("Non, Je suis ton père\n");
    return 0;
}
```

Question 1: Théorie

Donnez le résultat de l'exécution de ce programme (on suppose que l'appel « `execl` » est réussi). Expliquer qui affiche quoi et pourquoi.

Question 2: Arbre des processus

Soit le programe « `nemaxe` » suivant, où « `prog` » est un programme qui ne créer pas de processus.

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

int main(int argc, char*argv[]) {
    int retour;
    printf("%s\n", argv[0]); // A
    switch (fork()) { // B
        case -1 :
            perror("fork1()");
            exit(1);
        case 0 :
            switch (fork()) { // C
                case -1 :
                    perror("fork2()");
                    exit(1);
                case 0 :
                    if (execl("./prog","prog", NULL) == -1) {
                        perror("execl");
                        exit(1);
                    }
```

```
                    break;
                default :
                    exit(0);
            }
        default :
            wait(&retour);
        }
}
```

Représentez tous les processus créés par ce programme sous forme d'un arbre, en vous servant des lettres en commentaires. Quels sont les ordres possibles de terminaison des processus ?