4 - PROCESSUS UNIX

0. RAPPELS

L'appel système `fork()` crée un processus fils qui diffère de son père uniquement par ses numéros de `pid` et de `ppid`. `fork()` renvoie 0 pour le fils et le `pid` du fils créé pour le père. Juste après le `fork()`, les deux processus disposent des mêmes valeurs de données et de pile. Mais il n'y a pas de partage : chaque processus a sa propre copie.

L'appel système `wait()` bloque un processus en attente de la fin de l'exécution d'un fils qu'il a créé. `wait()` renvoie le numéro du fils qui vient de se terminer. On peut passer à `wait()` un paramètre de type `int*` qui permet de récupérer des informations sur la terminaison du fils.

Les appels système de la famille `exec` lancent un programme exécutable. Si l'appel système est réussi, le processus faisant le `exec` se termine lors de la fin du processus qu'il a lancé : on ne revient pas d'un `exec` réussi. Si l'appel échoue (et dans ce cas seulement), les instructions qui suivent le `exec` sont exécutées.

1. EXECUTION D'UN FORK SIMPLE

Soit le programme C suivant :

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

1.1.

Donnez les affichages effectués par le processus père et par le processus fils.

2. EXECUTION D'OPERATIONS FORK IMBRIQUÉES

Soit le programme suivant :

```
1:   int main(int argc, char *argv[]) {
2:
3:        int a, e;
4:
5:        a = 10;
6:        if (fork() == 0) {
7:             a = a *2 ;
8:             if (fork() == 0) {
9:                  a = a +1;
10:                 exit(2);
11:            }
12:            printf(" %d \n", a);
13:            exit(1);
```

```
14:       }
15:       wait(&e);
16:       printf("a : %d ; e : %d \n", a, WEXITSTATUS(e));
17:       return(0);
18:  }
```

2.1.

Donnez le nombre de processus créés, ainsi que les affichages effectués par chaque processus.

2.2.

On supprime la ligne 10, reprenez la question 2.1. en conséquence.

2.3.

Modifiez le programme initial pour créer un processus zombie pendant 30 secondes.

3. CREATION DE PROCESSUS EN CHAÎNE

On souhaite faire un programme qui crée une chaîne de processus telle que le processus initial (celui du main) crée un processus qui à son tour crée un second processus et ainsi de suite jusqu'à la création de N processus.

3.1.

Ecrivez ce programme.

3.2.

Modifiez le programme pour que le processus initial attende uniquement la fin de son fils.

3.3.

Modifiez le programme pour que le processus initial attende la fin de *tous* les processus créés.

4. MODIFICATION DU CODE EXÉCUTÉ : LA PRIMITIVE EXEC

On considère le programme suivant :

```
int main() {
    int p;
    p = fork();
    if (p == 0) {
       execl("/bin/echo", "echo", "je", "suis", "le", "fils", NULL);
    }
    wait(NULL);
    printf("je suis le pere\n");
    return EXIT_SUCCESS;
}
```

4.1.

Donnez le résultat de l'exécution de ce programme (on suppose que l'appel `execl` est réussi).

Soit le programme "nemaxe" suivant en C sous Unix, où "prog" est un programme qui ne crée pas de processus.

```
main(int argc, char*argv[]) {
   int retour;
   printf("%s\n", argv[0]);                                    // A
```

```
    switch (fork()) {
        case -1:
            perror("fork1()"); exit(1);
        case 0:// B
            switch (fork()) {
                case -1:
                    perror("fork2()");
                    exit(1);
                case 0:                                          // C
                    printf("ICI\n");
                    if (execlp("prog","prog",0) == -1) {
                        perror("execlp"); exit(1);
                    }
                    break;
                default:
                    exit(0);
            }
        default:
            wait(&retour);
    }
    printf("fin\n");
}
```

4.2.

Représentez tous les processus créés par ce programme sous forme d'un arbre, en vous servant des lettres en commentaires.

On change maintenant le nom de "nemaxe" par "prog" (celui de l'appel à execlp).

4.3.

Représentez le début de l'arbre de processus.