I2180 : Exercices de Programmation Système (2)

Les premiers exercices de cette séance ont pour but de vous faire comprendre l'appel système `fork`. Respectez scrupuleusement les consignes qui vous sont proposées et expliquez en détail les résultats obtenus, ne passez pas une étape sans vous être assuré qu'elle est bien comprise.

2.A. Exercice sur l'appel système : `fork`

a) Ecrivez un programme qui définit une variable « a » de type `int` et l'initialise à 5 ; crée un processus enfant ; affiche la valeur de retour de l'appel système `fork`. Ensuite, il définit dans l'enfant une nouvelle variable « b » de type `int`, lui affecte la valeur de « a * 2 » puis affiche le contenu de a et de b ; le parent définit une autre variable « b » de type `int`, lui affecte la valeur de « a * 5 » puis affiche le contenu de a et de b. Pour cet exercice, l'affichage peut se faire avec `printf`.

Qu'affiche ce programme lors de son exécution ?

b) Ecrivez un programme qui affiche la chaîne de caractères "trois .. deux .. un .." avant de créer un processus enfant qui affiche la chaîne de caractères "partez !". Les écritures doivent se faire grâce à l'appel système `write`.

Qu'affiche ce programme lors de son exécution ?

c) Ajoutez '\n' à la fin des deux chaînes de caractères.

Qu'affiche ce programme modifié lors de son exécution ?

d) Refaites les étapes b et c en remplaçant les appels système `write` par l'appel à la fonction `printf`.

Qu'affiche la première version du programme lors de son exécution ?

Qu'affiche ce programme si les chaînes de caractères se terminent par '\n' ?

e) Y-a-t-il une différence de comportement entre les deux versions ? Si oui, expliquez pourquoi puis modifiez ce qu'il faut pour régler le problème.

f) En conclusion, quelle est la différence qui vient d'être soulignée entre l'appel système `write` et la fonction `printf` ?

2.B. Exercice sur les appels système : `fork` et `wait`

a) Ecrivez un programme qui crée un processus enfant qui affiche la string "4 5 6\n" alors que le parent affiche la string "1 2 3\n".

Testez ce programme et expliquez le résultat obtenu.

b) Modifiez votre programme afin que l'enfant affiche toujours "4 5 6\n" avant que le parent n'affiche "1 2 3\n".

c) Modifiez votre programme afin que l'enfant se termine par un `exit` et que le parent vérifie si le pid de l'enfant récupéré via un "wait" est bien le pid du processus enfant créé par lui, puis affiche l'exit code de ce processus enfant. Consultez le man de `waitpid` pour trouver comment récupérer l'exit code du processus enfant.

2.C. Exercice sur les appels système : `fork` et `exec`

Vous allez écrire un programme interactif qui, au final :

- Saisira au clavier le nom d'un fichier script ;
- Ouvrira le fichier en l'écrasant (vidant son contenu) ou le créera s'il n'existe pas encore ;
- Rendra le fichier exécutable par son propriétaire si nécessaire, sans modifier ses autres permissions ;
- Exécutera un `ls -l` de ce fichier script via un `exec` ;
- Saisira au clavier, ligne par ligne, le contenu du script en l'écrivant en parallèle dans le fichier ;
- Exécutera ensuite un `cat` du contenu du script via un `exec` ;
- Exécutera finalement le script en question via un `exec`.

Suivez l'ordre des points **ci-dessous** en testant le caractère fonctionnel de votre code de temps en temps.

a) Ecrivez une fonction qui prend en arguments le retour d'un appel système et un message d'erreur. En cas d'erreur lors de l'appel système, la fonction imprime le message sur la sortie d'erreur puis fait un `exit(EXIT_FAILURE)`. Vous utiliserez cette fonction dans votre code chaque fois que cela sera approprié.

b) Incluez la fonction ci-dessous dans votre programme, afin de gérer les appels système de type `fork`. `*handler` y est un pointeur sur la fonction contenant le code à exécuter dans l'enfant. `fn` y représente le "file name" du fichier à traiter dans l'`execl`, chaque `execl` étant exécuté dans un handler dédié. Vous créerez autant de fonction de ce type que nécessaire par la suite.

```
static pid_t fork_and_run(void(*handler)(char*), char* fn){
      pid_t pid = fork();
      if(pid == -1){
            perror("Fork failed.\n");
            exit(EXIT_FAILURE);
      }
      if(pid == 0){
            (*handler)(fn);
            exit(EXIT_SUCCESS);
      }
      return pid;
}
```

c) Effectuez la lecture du nom du fichier script au clavier, à l'aide d'un appel système `read`, sans vérification particulière. Ouvrez le fichier en question en l'écrasant, ou créez-le si nécessaire.

d) A l'aide d'un `fork-exec`, effectuez un `ls -l` sur le nom du script.

e) A l'aide d'appels système `read` et `write`, lisez au clavier, ligne par ligne, le contenu du script en l'écrivant en parallèle dans le fichier créé.

f) A l'aide d'un `fork-exec`, effectuez un `cat` du contenu du script.

g) Libérez les ressources, puis exécutez le contenu du script à l'aide d'un `fork-exec` avant de terminer.

h) Lors de la saisie du nom du script, imposez un nom de maximum 20 caractères et gérez au mieux les fausses manœuvres de l'utilisateur.

i) Si le fichier script existe déjà et qu'il n'a pas la permission d'exécution par son propriétaire, rajoutez-la lui sans modifier ses autres permissions (faite appel aux appels systèmes `fstat` et `fchmod`).