Sommaire

Séquence 1 : l'appel fork

1) Expliquer, en utilisant un schéma, l'utilisation suivante de la primitive fork :

```
if (fork())
    printf("Je suis le père\n");
else
    printf("Je suis le fils\n");
```

```
$ man fork
..
En cas de succès, le PID du fils est renvoyé au processus parent, et 0
est renvoyé au processus fils. En cas d'échec -1 est renvoyé dans le
contexte du parent, aucun processus fils n'est créé, et errno contient
le code d'erreur.
```

2) Compléter (père/fils) l'affichage dans le programme suivant :

```
int main()
{
   int pid;
   pid = fork();
   if (pid == -1)
   {
      perror("fork"); exit(-1);
   }
   if (pid == 0) printf("Je suis le ...\n");
   if (pid != 0) printf("Je suis le ...\n");
}
```

3) Améliorer le programme en affichant les PID/PPID de chaque processus (voir les appels getpid() et getppid()) avec les messages suivants : "Je suis le père, mon PID est x et j'ai créé un fils dont le PID est x" et "Je suis le fls, mon PID est x et mon père a le PID x". Donner l'affichage obtenu.

Séquence 2 : généalogie de processus

4) Quel affichage produit le code suivant ? Combien de processus sont créés ? Justifier (un schéma peut être utile ;).

```
setbuf(stdout, NULL); // pour éviter la bufférisation sur l'affichage
printf("+");
fork();
printf("+");
fork();
printf("+");
```

Remarque : le \n force l'affichage sinon c'est bufférisé !

Pour vous aider, vous pouvez utiliser cet exemple :

```
printf("1\n");
fork();
printf("2\n");
fork();
printf("3\n");
```

5) Combien de processus sont créés ? Justifier (un schéma me semble indispensable). Quel affichage produit le code ci-dessous ?

```
setbuf(stdout, NULL);
for (i = 1; i <= 4; i++)
{
   printf("+");
   fork();
   printf("+");
}
```

Puis avec une variable compteur :

```
int i, compteur = 0 ;

setbuf(stdout, NULL);
for (i = 1; i <= 4; i++)
{
   printf("%d", compteur++);
   fork();
   printf("%d", compteur++);
}
```

Séquence 3 : les appels exec

Rappel : à partir d'un programme en C, il est possible d'exécuter des processus de plusieurs manières avec soit l'appel system soit les appels exec.

Exemple 1 : l'appel system()

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

int main()
{
   printf("Résultat de la commande ps fl :\n");
   system("ps fl");
   printf("Fin\n");
   return 0;
}
```

Exemple 2 : l'appel execlp()

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

int main()
{
   printf("Résultat de la commande ps fl :\n");
   execlp("ps", "ps", "fl", NULL); // cf. man execlp
   printf("Fin\n");
}
```

6) Comparer au niveau des PID/PPID l'exécution des deux exemples ci-dessus. Conclure.

Remarque : vous pouvez utiliser la commande pstree !

7) Dans l'exemple 2, le message "Fin\n" ne s'affiche pas. Pourquoi ?

8) Compléter le programme suivant pour qu'il lance une calculatrice (kcalc) et un éditeur de texte (kwrite) :

```
if (! fork()) { printf("Fils 1 : je lance kcalc !\n"); ... }
if (! fork()) { printf("Fils 2 : je lance kwrite !\n"); ... }
```

Remarque : pour un envirommenemnt GNOME, utiliser alors les programmes suivants : gcalctool pour la calcutrice et gedit pour l'éditeur de texte.

Séquence 4 : synchronisation des terminaisons

9) Qui de ces 2 processus se termine le premier, le père ou le fils ? Essayez plusieurs fois.

```
if (fork())
{
      printf("Je suis le père\n");
      printf("Fin du père\n");
}
else
{
      printf("Je suis le fils\n");
      printf("Fin du fils\n");
}
```

10) En utilisant la primitive **wait**, écrire un programme dans lequel le père se termine toujours après son fils.

11) Reprendre le programme qui lance kcalc et kwrite et le modifier pour que le processus père attende la fin de tous ses fils et qu'il affiche le code retour renvoyé par ses fils. (cf. man 2 wait)

Séquence 5 : état d'un processus

12) Tester et analyser le programme **status.c** fourni.

Lorsqu'un processus se termine, il reste dans un état intermédiaire Zombie (entre la vie et la mort) en attendant que son père lise son code retour avec wait().

Si son père n'est plus en vie, il est alors « adopté » par le processus **init** de PID numéro 1.

13) Écrire un programme C permettant de mettre en évidence cette situation.

Si son père n'a pas encore lu son code retour (par exemple si il se retrouve dans une boucle infinie !) mais qu'il est encore en vie, le processus fils sera marqué **Zombie**. Cet état est visible avec la commande ps (colonne STAT).

14) Écrire un programme C permettant de mettre en évidence cet état intermédiaire pour le fils.