TD2-3 et TP2-3 – Premiers pas avec les appels système

Objectif : se familiariser avec les appels systèmes pour la gestion des processus via les primitives fork, getpid, getppid, wait, waitpid, exit, sleep, pipe, ainsi qu'exec et ses dérivées.

Exercice 1 : appels systèmes pour la gestion des processus

1- **Utilisation de fork(), getpid() getppid() :** écrire un programme en C qui créé deux fils. Tous les processus (le père et les fils) devront indiquer chacun leurs numéros de processus, le numéro de leur père et la valeur renvoyée par la primitive de création de processus. Exécuter le programme plusieurs fois. Les valeurs affichées et l'ordre d'affichage sont-ils toujours les mêmes ?
2- **Utilisation de wait() :** modifier le programme C précédent afin que le père attende la fin d'exécution de ces deux fils avant de se terminer. Il affichera un message d'adieu juste avant de se terminer.
3- **Utilisation de sleep() :** modifier le programme C précédent afin que chacun des fils attende 5 secondes avant de se terminer. Chacun affichera un message avant et après l'attente.

Exercice 2 : Duplication de processus et partage des pointeurs de lecture ou d'écriture.

1- Ecrire en C un programme qui commence par un appel de fork() pour dupliquer le processus initial, le père, pour obtenir un fils. Le père et le fils ont ensuite le même comportement. Ils ouvrent le fichier "./test.data" puis y lisent 10 caractères et les affichent sur la sortie standard (écran) après avoir affiché leur identité. Effectuer ensuite le fork() juste avant la lecture. Répéter l'exécution plusieurs fois et déterminer quels sont les caractères lus par le processus père et le processus fils.
2- Dans le père, positionner le pointeur de lecture sur le 30ème caractère par appel de lseek() avant le fork(). Quelle est la position des caractères lus par le fils ensuite.
3- Modifier le programme pour que le fork() ait lieu avant l'ouverture du fichier. Comparer les résultats.

Exercice 3 :

Combien de processus le programme suivant crée-t-il ?

```
int main(int argc, char *argv[]) {
    fork();
    fork();
    fork();
    exit(0);
    }
```

Dessiner l'arbre des processus engendrés.

Exercice 4 : fork–bombe

Que fait le programme suivant ?

```
#define N 10
int main() {
```

```
int i = 1;
while (fork() == 0 && i <= N) i++;
printf("%d\n",i);        exit(0);
}
```

Exercice 5 : appels systèmes pour la gestion des processus

1- **Utilisation de pipe() et wait() :** modifier le processus de la question 1 de l'exercice 1 pour que le processus père puisse maintenant permettre à l'utilisateur de rentrer des nombres. Il transmet à l'aide d'un ou plusieurs tubes les nombres impairs au premier processus fils qu'il a créé et les nombres pairs au second (on transmettra au choix soit la représentation binaire de chaque entier, soit la suite de chiffres correspondante). Quand l'utilisateur rentre la valeur 0, le père envoie également la valeur 0 à ses fils pour leur donner l'ordre de se terminer, puis, attend qu'ils soient terminés tous les deux avant de se terminer. Chaque processus laisse un message d'adieux avant sa terminiaison.
2- **Utilisation de sleep () :** modifier le processus de la question 2 afin qu'à la réception de la valeur 0, le processus père attend une seconde et envoie à ses fils le message qui demande à tout le monde de se terminer.
3- **Utilisation de wait() et exit() :** Lorsque l'utilisateur rentre la valeur 0, le processus père ferme les tubes et attend la terminaison de ses fils. Chaque processus fils doit émettre un message d'adieux avant de se terminer et retourner la dernière valeur traitée. Le père affiche l'identité de ses fils et les valeurs qu'ils retournent.

Exercice 6 : utilisation d'execlp

Ecrire un programme en C qui propose à l'utilisateur le menu suivant jusqu'à ce que le choix "quitter" soit sélectionné :

1/ ls

2/ ps -ef

3/ find . -name data -print

4/ quitter