TP2

Exercice 1 : Héritage

Exécuter le programme suivant puis expliquer le résultat obtenu.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/times.h>
#include <unistd.h>

char buf[1024]; // pour récupérer le répertoire de travail
struct tms temps; // pour récupérer les nombres d'unités d'horloge par seconde : clics
int main(){
int i,valeur_fork;
nice(10); //augmentation de 10 de la valeur du nice
for (i=0;i<10000000;i++); // une boucle consommatrice de CPU
valeur_fork = fork();
if (valeur_fork==0) {
printf("caractéristiques du fils \n");
printf("fils :uid=%d euid= %d egid=%d\n ", getuid(), getegid());
printf(" fils : répertoire de travail : %s\n",getcwd(buf,1024));
printf("fils : nice : %d \n",nice(0)+20);
times(&temps);// en clock ticks
printf("fils : clics en mode utilisateur : %ld \n", temps.tms_utime);
printf("fils : clics en mode système : %ld \n", temps.tms_stime);
printf("fils : fin du processus\n\n");
}
else{
sleep(5); // pour partir après la terminaison du fils
printf("caractéristiques du père \n ");
printf("père : uid=%d euid= %d egid=%d\n ",getuid(),getegid());
printf("père : répertoire de travail : %s\n ",getcwd(buf,1024));
printf("père : nice : %d \n",nice(0)+20);
times(&temps);
printf("père : clics en mode utilisateur : %ld \n", temps.tms_utime);
printf("père : clics en mode système : %ld \n ", temps.tms_stime);
printf("père : fin du processus\n\n");
}
return 0; }
```

Solution :

En exécutant le programme ci-dessus, on peut bien vérifier les attributs que le processus fils hérite du processus père.

.**Remarque :**

La primitive *nice* : permet d'exécuter un programme avec une priorité

d'ordonnancement modifiée.
La primitive *getuid* : retourne l'identifiant de l'utilisateur qui a lancé le processus.
La primitive *getegid*: renvoie le numéro du groupe du propriétaire effectif du processus.
La primitive *getcwd* : donne le chemin du répertoire de travail courant.
La structure *tms* prédéfini dans le fichier sys/times : permet de calculer le CPU consommés. Cette structure accessible au moyen de la fonction *times*.
tms_utime : temps CPU écoulé en exécutant les instructions du processus appelant.
tms_stime : le temps CPU passé dans les fonctions système exécutées pour le processus appelant

Exercice 2: Les Processus Zombies

1) Exécuter le programme ci-dessous, puis tapez ps -la dans un autre terminal avant la fin du processus père, puis une autre fois avant la fin du processus fils. Quels sont les ppid du père et du fils ? Donnez une explication.
2) Changer les valeurs des deux variables *attente_fils* et *attente_pere* par 5, 20 respectivement, lancer le programme puis retaper la commande ps -la dans un autre terminal avant la fin du père, avant la fin du fils. Que constatez-vous ?

```
#include <stdio.h>
#include <sys/times.h>
#include <stdlib.h>
#include <unistd.h>
int main() {
int valeur_fork;
int attente_fils=20, attente_pere = 10;
switch(valeur_fork=fork()) {
case -1:
perror("fork error"); break;
case 0:
printf("Je suis processus fils avec PID = %d\n",getpid());
sleep(attente_fils);
printf("fils attente finie\n");
break;
default:
printf("Je suis processus père avec PID = %d\n",getpid());
sleep(attente_pere);
printf("père attente finie \n");
break;
}}
```

Solution :
Un processus zombie c'est un processus qui s'est achevé mais il dispose toujours d'un identifiant de processus (PID) et reste donc encore visible dans la table des processus actifs.