TP
FORK, EXECVE et DUP

Exercice n°1 – fork()

Soit le code suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main(int argc, char *argv[])
{
int PID;
printf("Debut du programme...\n");
PID=fork() ; //c'est à ce moment que les deux processus se séparent
if (PID > 0 )
{
//Ici on est avec le père
printf("Voici le processus pere. Le PID du processus est %d. Le PID de l'ancetre est
%d\n",getpid(),getppid());
sleep(2);
}

if ( PID == 0 )
{
//Ici avec le fils
printf("Voici le processus fils. Le PID du processus est %d. Le PID de l'ancetre est
%d\n",getpid(),getppid());
}

if ( PID < 0 ){
printf("Erreur lors de l'appel système");
return EXIT_SUCCESS;
}

}
```

1. A partir de quel moment le processus fils est-il créé ? Un nouveau descripteur de processus est-il créé ?
2. Quelle valeur de retour permet d'identifier le père du fils ?
3. Modifiez le code source pour que le processus fils créé à son tour un processus fils affichant un nouveau
message.
4. Quel est le rôle de la ligne de code sleep(2); ? Que se passe-t-il si elle n'est pas présente ?

Exercice n°2 – execve()

Soit le code suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main(int argc, char *argv[])
{
printf("Debut du programme...\n");
printf("On va essayer d'executer un autre programme si possible...");
//remplacez ci-dessous "/home/jean" par le chemin du fichier executable.
if ( execve("/home/jean/exemple.exe",NULL,NULL)==-1)
printf("et ben non...\n");
else
printf("Erreur lors de l'appel système");
return EXIT_SUCCESS;
}
```

Création de l'exécutable exemple.exe appelé par execve()

Tapez le code suivant dans un éditeur de texte et enregistrez le sous exemple.c :

```
#include <stdio.h>
int main ()
{
printf("Je suis le fichier exécuter par EXECVE. Bravo !\n");
return 0;
}
```

Dans un shell, depuis le répertoire où a été enregistré le fichier, tapez ensuite la commande suivante :

```
gcc exemple.c -o exemple.exe
```

Copiez/collez l'exécutable généré dans le répertoire du premier.

Modifiez votre projet pour que le fichier exemple.exe soit lancé par un processus fils créé grâce à un fork().

Modifiez encore une fois votre projet pour que le processus fils tourne en boucle une fois lancé (modifiez son code source).

Que se passe-t-il une fois que le processus père se termine ? Expliquez.

Exercice n°3 – dup()

Soit le code suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
int main(int argc, char *argv[])
{
int desc;
int NbCar;
char tab[200];
int val;
if ( (desc=open("texte.txt",O_RDONLY)) == NULL )
printf("Erreur lors de l'ouverture du fichier");
else
{
printf("Ouverture fichier OK\n");
close(0);
if ( (val=dup(desc)) == -1 )
{
printf("Erreur lors de duplication : %d \n",val);
}
else
{
printf("Duplication OK\n");
gets(tab);
printf("Données lues depuis le canal d'entrée standard : %s\n",tab);
}
close(desc);
}
return EXIT_SUCCESS;
}
```

Dans le dossier de votre code source, créez y un fichier texte.txt avec une phrase quelconque.

1. Quelles sont les données lues par le canal d'entrée standard ?
2. Quelle est l'effet du code close(0); ?
3. Quelle est la nouvelle source du canal d'entrée standard ?
4. Modifiez votre projet pour que le canal standard de sortie soit dans le fichier sortie.txt. A l'exécution, y a-t-il toujours un affichage à l'écran ? Expliquez.

Exercice 4 Appel système Wait()

En vous aidant de la documentation de l'appel système wait(), créez un programme qui lance 3 fils et qui signale la fin de chacun d'eux par un message de ce type « La fin d'un fils est détectée. Le processus de PID 2043 est terminé ». Afin de simuler un traitement, chaque processus fils attendra quelques secondes. A la fin le programme doit ajouter le message « Tous les fils ont terminé ».