TP PROCESSUS UNIX

Exercice 1

Après compilation du fichier exo1.c ci dessous, on exécutera le programme exo1 plusieurs fois. Pourquoi cinq lignes sont-elles affichées alors qu'il n'y a que 3 printf dans le programme ?
Ajouter maintenant la ligne suivante derrière l'appel à fork :
`if (valeur == 0) sleep (4);`
Que se passe-t-il ? Pourquoi ?

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

int main (){
  pid_t valeur;
  printf("printf-0 Avant fork : ici processus numero %d\n", (int)getpid());
  valeur = fork();
  printf("printf-1 Valeur retournee par la fonction fork: %d\n", (int)valeur);
  printf("printf-2 Je suis le processus numero %d\n", (int)getpid());
  return 0;
}
```

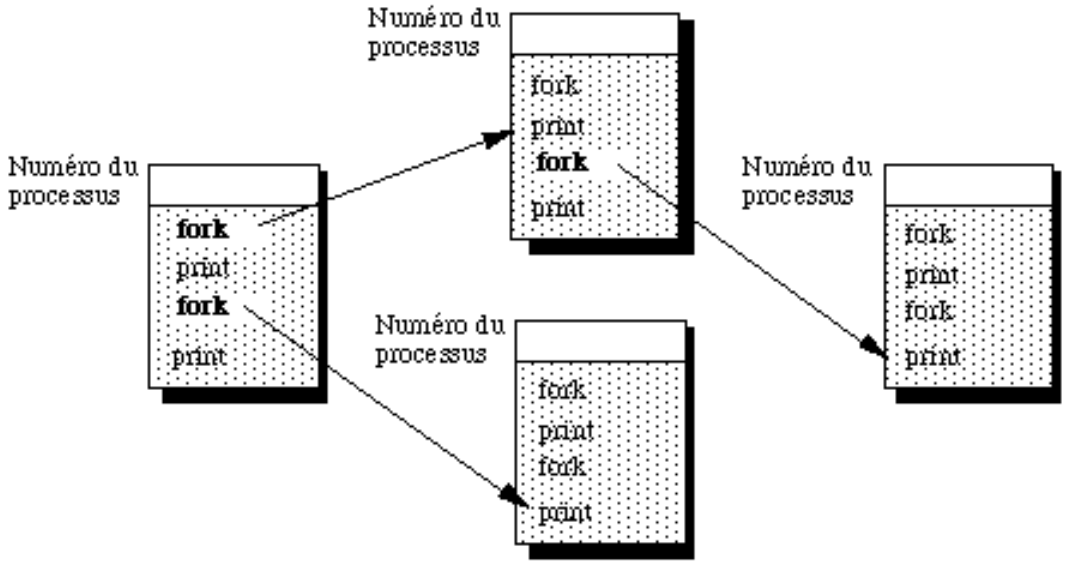
Exercice 2

Compiler fork-mul.c, puis l'exécuter.

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

int main () {
  pid_t valeur, valeur1 ;
  printf (" print 1 - Je suis le processus pere num=%d \n",(int)getpid() );
  valeur = fork();
  printf (" print 2 - retour fork: %d - processus num= %d -num pere=%d \n",
         (int)valeur, (int)getpid(), (int)getppid() );
  valeur1 = fork();
  printf (" print 3 - retour fork: %d - processus num= %d -num pere=%d \n",
         (int)valeur1, (int)getpid(), (int)getppid() );
  return 0;
}
```

Après exécution et à l'aide du schéma suivant, relever les numéros des processus et numéroter l'ordre d'exécution des instructions printf de façon à retrouver l'ordre d'exécution des processus.



Expliquer ce qui se passe si on relève un numéro de processus égal à 1.

Exercice 3

Compléter, compiler, exécuter et comprendre le programme suivant :

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

int main () {
  int valeur, ret_fils,etat ;
  printf ("Je suis le processus pere de pid %d \n", (int)getpid());
  valeur=fork();
  switch (valeur)
    {
    case 0 :
      printf
       ("\t\t\t\t********\n\t\t\t\t* FILS *       \n\t\t\t\t********\n");
      printf ("\t\t\t\tProc fils de pid %d \n\t\t\t\tPere de pid %d \n",
             (int) getpid(),(int) getppid() );
      printf("\t\t\t\tJe vais dormir 30 secondes ...\n");
      sleep (30);
      printf
           ("\t\t\t\tJe me reveille ,\n\t\t\t\tJe termine mon execution par un EXIT(7)\n");
        --- > ajout fin avec valeur de retour 7
    case -1:
      printf ("Le fork a echoue");
        --- > ajout fin avec valeur de retour 2

    default:
      printf("********\n* PERE *\n********\n");
      printf ("Proc pere de pid %d \nFils de pid %d \n",
             (int) getpid(),valeur );
      printf ("J'attends la fin de mon fils...\n");
        --- > ajout attente fin fils
      printf
       ("Mon fils de pid %d est termine,\nSon etat etait : %04x\n",
```

```
        ret_fils,etat);
    }
 return 0;
}
```

Exercice 4

Completer, compiler puis exécuter le programme suivant :

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

int main (int argc, char *argv[]){
  int Pid;
  int Fils,Etat;

  if (--→ test si 1 paramètre){
    printf(" Utilisation : %s fic. a executer ! \n", argv[0]);
    exit(1);
    }

  printf (" Je suis le processus %d je vais faire fork\n",(int)getpid());
  Pid=fork();
  switch (Pid){
    case  0 :
      printf (" Coucou ! je suis le fils %d\n", (int)getpid());
      printf (" %d : Code  remplace par %s\n", (int)getpid(), argv[1]);
      execl(--→ recouvrement par le programme passé en paramètre);
      printf (" %d : Erreur lors du exec \n", (int)getpid());
      exit (2);
    case   -1 :
      printf ("Le fork n'a pas reussi ");
      exit (3);
    default :
      /* le pere attend la fin du fils */
      printf (" Pere numero %d attend\n ",(int) getpid());

---→ ajout attente du fils

      printf ( " Le fils etait : %d ", Fils);
      printf (" ... son etat etait :%04x (hexa) \n",Etat);
      exit(0);
    }
  return 0;
}
```

Pour en vérifier le bon fonctionnement, on fera exécuter par execl un fichier très simple, par exemple celui dont le source est ci-dessous :

```
int main (void) {
   printf("Bonjour, ici %d !\n", (int)getpid() );
   sleep (4);
   return 6;
}
```